

版权注意事项：

- 1、书籍版权归作者和出版社所有
- 2、本PDF仅限用于个人获取知识，进行私底下的知识交流
- 3、PDF获得者不得在互联网上以任何目的进行传播
- 4、如觉得书籍内容很赞，请购买正版实体书，支持作者
- 5、请于下载PDF后24小时内删除本PDF。

Linux集群 和自动化运维

余洪春 著

Linux Cluster and Automation Operation

- 高级运维架构师、资深系统运维工程师十余年工作经验总结，基于一线运维工作提炼，从Linux集群、Python自动化运维和亿级PV网站架构设计等多角度讲解，以实践案例指导读者掌握到Linux系统集群和自动化运维技巧。
- 姊妹篇《构建高可用Linux服务器》被《程序员》杂志和51CTO等权威媒体评为“10大最具技术影响力的图书”和“最受读者喜爱的原创图书”。



机械工业出版社
China Machine Press

云计算是一种流行趋势，云计算中众多成熟产品的流行对于传统的运维知识体系也是一种冲击和挑战，笔者从事Linux系统运维、运维架构师的工作已有10多年，之前主要是从事传统Web运维相关工作，现在也随着趋势转到了云计算平台方向。对于运维人员来说，海量主机的运行维护、海量数据的分析和汇总，都是非常具有技术含量和挑战性的。许多读者朋友经常在交流中谈到从事系统运维工作3~5年以后就不知道如何继续学习和规划自己的职业生涯了。笔者希望通过此书，跟大家分享下自己的工作经验和心得，通过项目实践和线上环境案例，帮助大家迅速了解Linux运维人员的工作职责和方向，迅速进入工作状态且得到成长。本书最大的特点就是与实践紧密结合，所有理论知识、方法、技巧和案例均来自实际环境，书中内容涵盖了生产环境下的Shell和Python脚本、Puppet自动化运维及Python自动化运维、高可用Linux集群构建及亿级PV网站架构设计等主题。

Linux 集群和自动化运维 / 余洪春著. —北京: 机械工业出版社, 2016.8 (2016.12重印)
Linux/Unix 技术丛书
ISBN 978-7-111-24138-8

I. L… Ⅱ.余… Ⅲ.Linux 操作系統 IV.TP316.89

中國版本圖書館CIP數據核字(2016)第178052號

Linux集群 和自动化运维

Linux Cluster and Automation Operation

余洪春 著



机械工业出版社
China Machine Press

图书在版编目 (CIP) 数据

Linux 集群和自动化运维 / 余洪春著. —北京: 机械工业出版社, 2016.8 (2016.12 重印)
(Linux/Unix 技术丛书)

ISBN 978-7-111-54438-8

I. L… II. 余… III. Linux 操作系统 IV. TP316.89

中国版本图书馆 CIP 数据核字 (2016) 第 176055 号

Linux 集群和自动化运维

出版发行: 机械工业出版社 (北京市西城区百万庄大街 22 号 邮政编码: 100037)

责任编辑: 陈佳媛 杨绣国

责任校对: 董纪丽

印 刷: 北京市荣盛彩色印刷有限公司

版 次: 2016 年 12 月第 1 版第 2 次印刷

开 本: 186mm×240mm 1/16

印 张: 23.25

书 号: ISBN 978-7-111-54438-8

定 价: 79.00 元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

客服热线: (010) 88379426 88361066

投稿热线: (010) 88379604

购书热线: (010) 68326294 88379649 68995259

读者信箱: hzit@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问: 北京大成律师事务所 韩光 / 邹晓东

Foreword 推荐序一

在全球“互联网+”的大背景下，互联网创业企业的数量如雨后春笋般大量产生并得到了快速发展！对“互联网+”最有力的支撑就是 Linux 运维架构师、云计算和大数据工程师，以及自动化开发工程师等！

但是，随着计算机技术的发展，企业对 Linux 运维人员的能力要求越来越高，这就使得很多想入门运维的新手不知所措，望而却步，甚至努力了很久却仍然徘徊在运维岗位的边缘；而有些已经工作了运维人员也往往是疲于奔命，没有时间和精力去学习企业所需的新知识和技能，从而使得个人的职业发展前景大大受限。

本书就是在这样的背景下诞生并致力于为上述问题提供解决方案的，本书是作者余洪春先生 10 多年来一线工作经验的“再”结晶，此前作者已经出版过 Linux 集群方向的图书（《构建高可用 Linux 服务器》），本次出版的书是作者对运维行业的再回馈。

书中不仅涵盖了入门运维人员必须了解的 IDC 和 CDN 服务的选型、Linux 系统及常见服务的优化实践内容，还有对于企业运维人员需要的大规模集群场景下必备的运维自动化 Shell 和 Python 企业开发应用实践案例、热门的自动化运维工具的企业应用实践、大规模集群及高可用的企业案例分享与安全防护等。

本书能够帮助运维人员掌握业内运维实战专家的网站集群的企业级应用经验的精髓，从而以较高的标准胜任各类企业运维的工作岗位，并提升自己的运维职业发展竞争力，值得一读！

——老男孩 老男孩 Linux 实战运维培训中心总裁

《跟老男孩学 Linux 运维：Web 集群实战》作者

读者对象

本书适合以下读者阅读

□ 中高级系统管理员

□ 系统架构设计师

推荐序二 *Foreword*

本书作者余洪春先生和我相识于 ChinaUnix 举办的一次技术交流活动——“千万级 PV 高性能高并发网站架构与设计交流”，当时他已经在宣传自己的第一本著作——《构建高可用 Linux 服务器》，该书凝聚并整合了他多年来在一线工作的经验结晶，以至时至今日，该书仍是一本在国内非常经典的运维原创著作，现在已经更新到第三版，这种对技术不断进行完善的坚持及工匠精神让我深深折服。这次能受邀为他的新书《Linux 集群和自动化运维》写推荐序，让我倍感荣幸。

本书覆盖了 Linux 集群服务的核心技术，同时还介绍了基于 Python 语言构建的主流自动化运维工具，包括 Puppet、Fabric、Ansible 等，这些都是 DevOps 工具元素周期表中最闪亮的内容，也是运维人员必备的技能。本书中分享的案例是余洪春先生多年实战经验的精华，具有非常高的参考价值及借鉴意义。

书中内容从互联网业务平台构建及自动运维的场景出发，以常见的业务服务为基础，给出了大量的实战案例，这些都是作者在十余年的互联网运维工作中总结出来的宝贵经验，相信会给读者带来不少启发及思考。

更难能可贵的是，作者能从通俗易懂的角度出发，由浅入深地剖析自动运维管理之道。对于不同水平层次的读者来说，都能有效地阅读和吸收，也能根据实际需要各取所需。

最后，感谢余洪春给中国互联网从业者带来这么好的图书，我相信阅读本书的每一位读者都能从中获取提升的能量，为企业及行业做出自己的贡献。

腾讯高级工程师 刘天斯

Preface 前言

为什么要写这本书

笔者从事系统运维和运维架构师的工作已有 10 多年，现在在一家外企担任云平台运维架构师的职位。云计算是现在的主流技术，未来也有很好的发展趋势，云计算的流行对于传统的运维知识体系来说，其实也造成了冲击，有很多读者经常向笔者咨询工作中的困惑，比如从事系统运维工作 3~5 年后就不知道该如何继续学习和规划自己的职业生涯了。因此笔者想通过此书，跟大家分享一下自己的工作经验和心得（包括传统运维和云平台运维工作的区别与对比），以期解决大家在工作中的困惑。本书提供了大量项目实践和线上案例，希望能让大家迅速了解 Linux 运维人员的工作职责，快速进入工作状态并找到成长方向。希望大家通过阅读此书，能够掌握 Linux 系统集群和自动化运维及网站架构设计的精髓，从而能够轻松愉快地工作，并提升自己的职业技能，这就是笔者写作此书的初衷。

运维架构师之路

在成为运维架构师之前，笔者从事过很长一段时间的系统集成、运维和管理工作，在 CDN 门户网站、电子商务领域也有不少的沉淀和积累，在之前的《构建高可用 Linux 服务器》一书中已经跟大家分享了很多跟 Linux 集群有关的知识。笔者目前的主要工作职责是维护和优化公司的 DSP 电子广告业务平台，主要方向是云计算和大数据方面。需要维护的数据中心和机器数量非常之多，所以自动化运维和 DevOps 是目前的主要工作方向，此外，也会涉及网站架构设计及调优工作，因此在此书中特意将这部分工作经验分享出来，希望大家能从中学到新的知识体系，借以提升自己的职业技能。

读者对象

本书适合以下读者阅读。

- ☐ 中高级系统管理员
- ☐ 系统架构设计师

- 高级程序开发人员
- 运维开发工程师

如何阅读本书

本书是笔者对实际工作中积累的技术和经验所做的总结，涉及大量的知识点和专业术语。全书总共分为三大部分，第一部分包含第 1 章和第 2 章，主要讲解进行系统架构设计的软硬件环境，以及生产环境下的 Shell 脚本和 Python 脚本。其中，第 2 章的内容是以 Shell 为主，Python 为辅，Shell 部分讲得比较详细，Python 部分需要重点关注的地方也有所提及。之所以这样安排，主要是考虑到大多数搞开发的读者或 DevOps 工程师都是 Java 程序员出身，对 Shell 脚本语言不是很熟悉。第二部分包含第 3 章、第 4 章和第 5 章，主要讲自动化运维，包括 Fabric、Ansible 和 Puppet 三大工具，大家可以结合自己的实际环境来选择对应的工具。第三部分包含第 6 章、第 7 章和第 8 章，主要讲的是 Linux 集群和网站架构设计，特别是第 8 章，分别以百万 PV、千万 PV 及亿级 PV 的网站为例来详细说明网站系统架构设计的相关技术，然后细分五层来解说网站的架构，并指出了设计网站的压力及关注点所在。

大家可以根据自己的职业发展和工作需求来选择不同的章节进行阅读或学习。

关于本书中的配置文件、Shell 脚本和 Python 脚本的编号，这里也略作说明，比如 1.5.3 节中有 1.sh，表示这是 1.5.3 节的第一个 Shell 脚本；如果是 2.py，则表示是 1.5.3 节的第二个 Python 脚本；其他依此类推，在哪个章节中出现的配置文件或脚本就在哪个章节中寻找，这样对照起来阅读理解会比较方便。此外，书中多次出现的 Nginx 配置文件 nginx.conf 也在对应的章节里。本书相关的 GitHub 地址为 <https://github.com/yuhongchun/automation>。

勘误

尽管笔者花费了大量的时间和精力来核对文件和语法，但书中难免还会存在一些错误和纰漏，如果大家发现有任何问题，都请及时反馈给我，相关信息可以发到个人邮箱 yuhongchun027@gmail.com。尽管无法保证对于每一个问题都会有一个正确答案，但我肯定会努力回答并且指出一个正确的方向。

致谢

感谢爱女媛媛的出生，你的降临是上天赐给我的最好礼物，是我进行写作的源泉和动力。

感谢我的家人，他们在生活上对我的照顾无微不至，让我有更多的精力和动力去工作和创作。

感谢好友三宝这么多年来对我的信任和支持，从始至终一直都在支持和信任我。

感谢机械工业出版社华章公司的编辑杨福川和杨绣国，在你们的信任、支持和帮助下，我才能如此顺利地全部书稿。

感谢好友老男孩和刘天斯，闲暇之余和你们一起交流开源技术和发展趋势，也是一种享受。

感谢 Linux 之父——Linus Torvalds，他不仅创造了 Linux 系统，而且还创造了 Git 这么神奇的版本管理软件。

余洪春（抚琴煮酒）

中国，武汉

目 录 Contents

推荐序一	1.5.1 服务器物理硬件的优化	25
推荐序二	1.5.2 利用 tuning-primer 脚本来	
前 言	调优 MySQL 数据库	25
第 1 章 系统架构设计的构建基础	1.6 小结	28
1.1 网站架构设计相关术语	第 2 章 生产环境下的 Shell 和 Python	
1.1.1 什么是 HTTP 1.1	脚本	29
1.1.2 什么是 Web 2.0	2.1 Shell 和 Python 语言的简单介绍	29
1.1.3 软件开发 C/S 结构与 B/S 结构的	2.2 Shell 编程基础	30
区别	2.2.1 Shell 脚本的基本元素	30
1.1.4 评估网站性能的专业术语	2.2.2 Shell 特殊字符	31
1.2 IDC 机房的选择及 CDN 的选型	2.2.3 变量和运算符	31
1.3 如何根据服务器应用选购服务器	2.3 Shell 中的控制流结构	42
1.4 CentOS 6.4 x86_64 最小化安装后的	2.4 sed 的基础用法及实用示例	45
优化	2.4.1 sed 的基础语法格式	46
1.4.1 系统的基础优化	2.4.2 sed 的用法示例	51
1.4.2 优化 Linux 下的内核 TCP 参数	2.5 awk 的基础用法及实用示例	56
以提高系统性能	2.6 生产环境下的 Shell 和 Python 脚本	
1.4.3 CentOS 6.4 x86_64 系统最小化	分类	61
优化脚本	2.6.1 备份类脚本	62
1.4.4 Linux 下 CPU 使用率与机器负载	2.6.2 统计类脚本	66
的关系与区别	2.6.3 监控类脚本	69
1.5 MySQL 数据库的优化		

2.6.4 开发类脚本	72
2.6.5 自动化类脚本	78
2.7 小结	80

第3章 轻量级自动化运维工具 Fabric

详解

3.1 Python 语言的应用领域	81
3.2 选择 Python 的原因	83
3.3 Python 的版本说明	83
3.4 增强的交互式环境 IPython	84
3.5 Python(x,y) 介绍	85
3.6 轻量级自动化运维工具 Fabric 介绍	86
3.6.1 Fabric 的安装	87
3.6.2 命令行入口 fab 命令详解	88
3.6.3 Fabric 的核心 API	88
3.7 Fabric 应用实例	92
3.7.1 开发环境中的 Fabric 应用实例	92
3.7.2 线上环境中的 Fabric 应用实例	93
3.8 小结	96

第4章 自动化部署管理工具 Ansible

简介

4.1 YAML 语言介绍	99
4.2 Ansible 的安装步骤	101
4.3 利用 ssh-keygen 设置 SSH 无密码 登录	105
4.4 Ansible 常用模块介绍	107
4.5 playbook 介绍	121
4.6 角色	126
4.7 Jinja2 过滤器	132
4.8 小结	136

第5章 自动化配置管理工具 Puppet

5.1 Puppet 的基本概念及介绍	137
---------------------------	-----

5.1.1 Puppet 简介	137
5.1.2 学习 Puppet 应该掌握 Ruby 基础	138
5.1.3 Puppet 的基本概念及工作流程 介绍	138
5.2 安装 Puppet 前的准备工作	140
5.3 Puppet 的详细安装步骤	141
5.4 Puppet 的简单文件应用	145
5.5 Puppet 的进阶操作	152
5.5.1 如何同步 Puppet-Client 端上的 常用服务	152
5.5.2 如何在 Puppet-Client 端自动安装 常用的软件包	153
5.5.3 如何自动同步 Puppet-Client 端 的 yum 源	153
5.5.4 如何根据不同名字的节点机器 推送不同的文件	155
5.5.5 如何根据节点机器名选择性地 执行 Shell 程序	158
5.5.6 如何快速同步 Puppet-Server 端的 www 目录文件	160
5.5.7 如何利用 ERB 模板来自动 配置 Apache 虚拟主机	165
5.5.8 如何利用 ERB 模板来自动 配置 Nginx 虚拟主机	168
5.6 Puppet 的负载均衡方式	172
5.7 用 GitHub 来管理 Puppet 配置 文件	173
5.8 小结	176

第6章 Linux 防火墙及系统安全篇

6.1 基础网络知识	177
------------------	-----

6.1.1	OSI 网络参考模型	177		介绍和应用	218
6.1.2	TCP/IP 三次握手的过程详解	178	6.11	工作中的 Linux 防火墙总结	220
6.1.3	Socket 应用及其他基础网络知识	181	6.12	Linux 服务器基础防护知识	221
6.2	Linux 防火墙的概念	182	6.13	Linux 服务器高级防护知识	222
6.3	Linux 防火墙在企业中的应用	183	6.14	如何防止入侵	222
6.4	Linux 防火墙的语法	184	6.15	小结	223
6.5	iptables 的基础知识	188	第 7 章 Linux 集群及项目案例分享		224
6.5.1	iptables 的状态 state	188	7.1	负载均衡高可用核心概念及常用软件	224
6.5.2	iptables 的 conntrack 记录	190	7.1.1	什么是负载均衡高可用	224
6.5.3	关于 iptables 模块的说明	191	7.1.2	以 F5 BIG-IP 作为负载均衡器	225
6.5.4	iptables 防火墙初始化的注意事项	192	7.1.3	以 LVS 作为负载均衡器	226
6.5.5	如何保存运行中的 iptables 规则	192	7.1.4	以 Nginx 作为负载均衡器	230
6.6	如何流程化编写 iptables 脚本	193	7.1.5	以 HAProxy 作为负载均衡器	231
6.7	学习 iptables 应该掌握的工具	196	7.1.6	高可用软件 Keepalived	232
6.7.1	命令行的抓包工具 TCPDump	196	7.1.7	高可用软件 Heartbeat	233
6.7.2	图形化抓包工具 Wireshark	197	7.1.8	高可用块设备 DRBD	233
6.7.3	强大的命令行扫描工具 Nmap	200	7.1.9	四、七层负载均衡工作流程对比	235
6.8	iptables 简单脚本: Web 主机防护脚本	203	7.2	负载均衡关键技术	237
6.9	线上生产服务器的 iptables 脚本	204	7.2.1	什么是 Session	237
6.9.1	安全的主机 iptables 防火墙脚本	205	7.2.2	什么是 Session 共享	237
6.9.2	自动分析黑名单及白名单的 iptables 脚本	207	7.2.3	什么是会话保持	238
6.9.3	利用 recent 模块限制同一 IP 的连接数	210	7.3	负载均衡器的会话保持机制	239
6.9.4	利用 DenyHosts 工具和脚本来防止 SSH 暴力破解	214	7.3.1	LVS 的会话保持机制	239
6.10	TCP_Wrappers 应用级防火墙的		7.3.2	Nginx 负载均衡器中的 ip_hash 算法	244
			7.3.3	HAProxy 负载均衡器的 source 算法	244
			7.3.4	服务器健康检测技术	249
			7.4	Linux 集群的项目案例分享	250

7.4.1 案例分享一：用 Nginx+Keepalived 实现在线票务系统	250
7.4.2 案例分享二：企业级 Web 负载 均衡高可用之 Nginx+Keepalived	253
7.4.3 案例分享三：Nginx 主主负载 均衡架构	265
7.4.4 案例分享四：生产环境下的高 可用 NFS 文件服务器	270
7.4.5 案例分享五：生产环境下 的 MySQL DRBD 双机高可用	280
7.4.6 案例分享六：生产环境下 的 MySQL 数据库主从同步	293
7.4.7 案例分享七：HAProxy 双机高可 用方案之 HAProxy+Keepalived	303
7.4.8 案例分享八：巧用 DNS 轮询做 负载均衡	308
7.5 软件级负载均衡器的特点介绍与 对比	313
7.6 网站系统架构设计图	315
7.7 小结	316

第 8 章 浅谈网站系统架构设计

8.1 网站架构设计规划预案	318
8.1.1 利用经验，合理设计	318
8.1.2 规划好网站未来的发展	319
8.1.3 合理选用开源软件方案	319
8.1.4 机房及 CDN 选型	319
8.1.5 节约成本	320
8.1.6 安全备份	320
8.2 百万级 PV 高可用网站架构设计	321
8.3 千万级 PV 高性能高并发网站 架构设计	323
8.4 亿级 PV 高性能高并发网站架构 设计	327
8.5 细分五层解说网站架构	333
8.6 小结	335

附录 A HAProxy 1.4 的配置文档

附录 B rsync 及 inotify 在工作中的应用

附录 C 用 Supervisor 批量管理进程

网站实施的初期就做好项目的成本预算和风险评估，并对系统的高可用及扩展性进行细数权衡，这些都是其工作职责所在。当然，在了解上述这些之前，首先应该了解一些网站架构设计相关的专业术语，下面我们就一起来看看。

1.1 网站架构设计相关术语

1.1.1 什么是 HTTP 1.1

HTTP 1.1 (Hypertext Transfer Protocol Version 1.1)，即超文本传输协议 - 版本 1.1，跟版本 1.0 是有区别的。

针对 HTTP 1.0 中 TCP 连接不能重复使用的问题，HTTP 1.1 采用了效率更高的持续连接机制，即客户端和服务器建立一次连接以后，后续相关的 HTTP 请求可以重复利用已经建立起来的 TCP 连接。

HTTP 1.1 是用来在 Internet 上传递超文本的传输协议。它是超文本传输协议 (HTTP) 协议族之

系统架构设计的构建基础

作为一名系统架构设计师，会面临着很多系统方面的架构设计工作，比如电子商务系统、CDN（内容分发网络）大型电子广告平台和 DSP 电子广告系统的运维方案确定及平台架构设计等，此外，还会涉及核心业务的系统在线优化及升级等工作。在以上这些工作中，又将包括多项选择：比如是考虑自建 CDN，还是租赁 CDN 系统；公司的业务系统所在的机房是考虑自建机房、托管机房，还是云计算平台，而选择托管机房，又会有更多的细节需要考虑，比如是选择电信机房、双线机房还是 BGP 机房，服务器应该如何选型，选择哪种操作系统等，这个时候系统架构设计师的经验和作用就体现出来了。他们应该在系统网站实施的初期就做好项目的成本预算和风险规避，并对系统的高可用及扩展性进行细致权衡，这些也是其工作职责所在。当然，在了解上述这些之前，首先应该了解一些网站架构设计相关的专业术语，下面就一起来看看。

1.1 网站架构设计相关术语

1.1.1 什么是 HTTP 1.1

HTTP 1.1（Hypertext Transfer Protocol Version 1.1），即超文本传输协议 - 版本 1.1，跟版本 1.0 是有区别的。

针对 HTTP 1.0 中 TCP 连接不能重复利用的情况，HTTP1.1 采用了效率更高的持续连接机制，即客户端和服务端建立 TCP 连接以后，后续相关联的 HTTP 请求可以重复利用已经建立起来的 TCP 连接。

HTTP 1.1 是用来在 Internet 上传送超文本的传送协议。它是运行在 TCP/IP 协议族之

上的 HTTP 应用协议，它可以使浏览器更加高效，并减少网络传输。任何服务器除了包括 HTML 文件以外，都还有一个 HTTP 驻留程序，用于响应用户请求。如果浏览器是 HTTP 客户，在向服务器发送请求时，向浏览器中输入一个开始文件或点击一个超级链接，浏览器就向服务器发送 HTTP 请求，此请求被送往由 URL 指定的 IP 地址。驻留程序接收到请求，在进行必要的操作后就会回送所要求的文件。

HTTP 1.1 支持持续连接。通过这种连接，就有可能在建立一个 TCP 连接后，发送请求并得到回应，然后发送更多的请求并得到更多的回应。由于把建立和释放 TCP 连接的开销分摊到了多个请求上，因此对于每个请求而言，由于 TCP 连接而造成的相对开销就被大大地降低了。而且，还可以发送流水线请求，也就是说在发送请求 1 的回应到来之前就发送请求 2。也可以认为，一次连接发送多个请求，由客户机确认是否关闭连接，而服务器会认为这些请求分别来自于不同的请求。

1.1.2 什么是 Web 2.0

Web 2.0，指的是利用 Web 的平台，由用户主导而生成内容的互联网产品模式，为了区别于网站雇员主导生成内容的传统网站而定义为 Web 2.0。Web 1.0 的盈利模式都基于一个共同点，即巨大的点击流量，无论是早期融资还是后期获利，依托的都是众多的用户和点击率，以点击率为基础融资上市或开展增值服务，充分体现了互联网的眼球经济色彩，例如早期的新浪、搜狐和网易等。

Web 2.0 是资源平等的体现。Web 2.0 的应用可以让人了解到目前万维网正在进行的一场改变——从一系列网站到一个成熟的、为最终用户提供网络应用的服务平台。这种概念的支持者期望 Web 2.0 服务在很多用途上能最终取代桌面计算机应用。虽然 Web 2.0 并不是一个技术标准，但是它包含了技术架构及应用软件。它的特点是鼓励信息的最终利用者通过分享，使得可供分享的资源变得更加丰富；相反的，过去网上的各种分享方式则显得支离破碎。

Web 2.0 是网络运用的新时代，网络成为了新的平台，内容因为每位用户的参与而产生，参与所产生的个人化内容，借由人与人（P2P）的分享，形成了现在的 Web 2.0 世界。

Web 2.0 的主要特点和基于这些特点所产生的具有代表性的服务如下。

1. 博客

博客（Blog）是 Web 2.0 最早期的服务之一，可使任何参与者拥有自己的专栏，成为网络内容的产生源，进而形成微媒体，为网络提供文字、图片、声音或视频信息。

2. 内容源

内容源（RSS）是伴随博客产生的简单文本协议，将博客产生的内容进行重新格式化输出，从而将内容从页面中分离出来，便于同步到第三方网站或提供给订阅者进行阅读。

3. Wiki

是一个众人协作的平台，方便编写百科全书、词典等。Wiki 指的是一种超文本系统，

这种超文本系统支持面向社区的协作写作，例如百度百科和维基百科。

4. 参与评论与评分的 Digg 机制

Web 2.0 最显著的特点之一是分享机制和去中心化，Digg 机制为更多的网络用户提供了参与网络建设的机制，无须进行内容贡献或创作，只要用户对网络内容进行评分或点评，即可参与到网络内容的建设过程当中。

5. 美味书签

美味书签 (Delicious) 不同于个人博客，用户可根据自己的喜好进行网络内容的收藏与转载，并将自己的收藏或转载整理成列表，分享给更多的用户，从而在网络上起到信息聚合与过滤的作用。

6. 社会化网络

社会化网络 (SNS) 从原有的以网站、内容为中心，转为侧重于以人与人之间的关联为中心，网络上每一个节点所承载的不再是信息，而是以具体的自然人为节点，形成的新型互联网形态。

7. 微博

微博 (Microblog) 作为博客的精简版，有较为严格的字数限制和政治立场限制。有更简单的发布流程和更随意 (被限制的话题、领域) 的写作方式，使得参与到网络内容贡献中的门槛降低，更大程度地推动了网络内容建设和个体信息贡献。

8. 基于位置信息的服务

基于位置信息的服务 (LBS) 是集地理信息系统 (GIS)、微博 (Twitter) 和移动设备 (Mobile) 及 A-GPS 定位服务于一体的增强型微博系统，其主导思想是每一条信息除了利用时间为索引，还加入了地理经纬度的索引，从而实现不仅可以通过时间对信息进行筛选，还可以利用地理坐标对信息进行合理的筛选。

9. 即时通信

即时通信 (IM) 软件可以说是目前中国网上用户使用率最高的软件。聊天一直是网民们上网的主要活动之一，网上聊天的主要工具已经从初期的聊天室、论坛变为以 QQ 和 Skype 为主要代表的即时通信软件。

1.1.3 软件开发 C/S 结构与 B/S 结构的区别

C/S 结构是大家熟知的软件系统体系结构，即 Client/Server (客户机 / 服务器) 结构，它通过将任务合理地分配到 Client 端和 Server 端，来降低系统的通信开销，不过需要安装客户端才可进行管理操作。B/S 结构，即 Browser/Server (浏览器 / 服务器) 结构，是随着 Internet 技术的兴起，对 C/S 结构的一种变化或改进的结构。在这种结构下，用户界面可完全通过 WWW 浏览器来实现。像 QQ、Skype 这类即时通信软件就属于 C/S 结构；而像百度、

Google 这样的搜索引擎就属于 B/S 结构。

随着计算机技术的不断发展与应用, 计算模式从集中式转向了分布式, 尤为典型的是 C/S 结构。两层结构 C/S 模式, 在 20 世纪 80 年代及 90 年代初得到了大量的应用, 最直接的原因是可视化开发工具的推广。之后, 它开始向三层结构发展。近年来, 随着网络技术的不断发展, 尤其是基于 Web 的信息发布和检索技术、Java 计算技术及网络分布式对象技术的飞速发展, 导致了很多应用系统的体系结构从 C/S 结构向更加灵活的多级分布结构演变, 使得软件系统的网络体系结构跨入一个新阶段, 即 B/S 体系结构。基于 Web 的 B/S 模式其实也是一种客户机/服务器模式, 只不过它的客户端是浏览器。为了区别于传统的 C/S 模式, 才特意将其称为 B/S 模式的。了解这些结构的特征, 对于系统的选型而言是很关键的。

下面是 C/S 结构与 B/S 结构的特点分析。

1. 系统的性能

在系统的性能方面, B/S 占有优势的是其异地浏览和信息采集的灵活性。任何时间、任何地点、任何系统, 只要可以使用浏览器上网, 就可以使用 B/S 系统的终端。

不过, 采用 B/S 结构时, 客户端只能完成浏览、查询、数据输入等简单功能, 绝大部分工作由服务器承担, 这就使得服务器的负担很重。采用 C/S 结构时, 客户端和服务器端都能够处理任务, 这虽然对客户机的要求较高, 但因此可以减轻服务器的压力。而且, 由于客户端使用浏览器, 使得网上发布的信息必须是以 HTML 格式为主, 其他格式的文件则多半是以附件的形式来存放的。而且 HTML 格式文件 (也就是 Web 页面) 不便于编辑修改, 给文件的管理带来了许多不便。

2. 系统的开发

C/S 结构是建立在中间件产品基础之上的, 要求应用开发者自己去处理事务管理、消息队列、数据的复制和同步、通信安全等系统级的问题。这对应用开发者提出了较高的要求, 而且还会迫使应用开发者投入很多精力来解决应用程序以外的问题, 这使得应用程序的维护、移植和互操作变得复杂。如果客户端是在不同的操作系统上, 那么 C/S 结构的软件还需要开发不同版本的客户端软件。

但是, 与 B/S 结构相比, C/S 技术的发展历史更为“悠久”。从技术成熟度及软件设计、开发人员的掌握水平来看, C/S 技术应是更成熟、更可靠的。

3. 系统的升级维护

C/S 系统的模块中只要有一部分发生改变, 就会关联到其他模块的变动, 这会使得系统的升级成本比较高。B/S 与 C/S 处理模式相比, 则大大简化了客户端, 只要客户端机器能上网就可以。对于 B/S 而言, 开发、维护等几乎所有的工作也都集中在服务器端, 当企业对网络应用进行升级时, 只需更新服务器端的软件就可以了, 这就降低了异地用户进行系统维护与升级的成本。如果客户端的软件系统升级比较频繁, 那么 B/S 架构的产品优势就更明显——所有的升级操作只需要针对服务器进行即可, 这对那些点多面广的应用是很有价

值的,例如一些招聘网站就需要采用 B/S 模式,客户端分散,且应用简单,只需要进行简单的浏览和少量信息的录入即可。

在系统安全维护上,B/S 则略显不足,B/S 结构尤其得考虑数据的安全性和服务器的安全性,毕竟现在的网络安全系数并不高。以 OA(办公自动化)软件为例,B/S 结构要实现办公协作过程中复杂的工作流控制与安全性控制,还有很多技术上的难点。因此,当前虽然出现了 B/S 结构的 OA 系统产品,但尚未大范围推广。

1.1.4 评估网站性能的专业术语

1. PV

PV(Page View)即访问量,中文翻译为页面浏览,即页面浏览量或点击量,用户每刷新一次就会被计算一次。PV 的具体度量方法就是从浏览器对网络服务器发出一个请求(Request),网络服务器接到这个请求后,会将该请求对应的一个网页(Page)发送给浏览器,从而产生一个 PV。在这里只要是请求发送给了浏览器,无论这个页面是否完全打开(下载完成),那么都被计为 1 个 PV。PV 反映的是浏览某网站的页面数,所以每刷新一次也算一次,也就是说 PV 与 UV(独立访客)的数量成正比,但 PV 并不是页面的来访者数量,而是网站被访问的页面数量。

2. UV

UV(Unique Visitor)即独立访问,访问网站的一台电脑客户端为一个访客,如果以天为计量单位,程序会统计 00:00 至 24:00 时间段内的电脑客户端。相同的客户端只被计算一次。一个电脑客户端可能有多个不同的自然人访问,但也只记一个 UV,通过不同的技术方法来记录,实际会有误差。如果企业内部通过 NAT 技术共享上网,那么出去的公网 IP 有且只有一个,这个时候在程序里面统计的话,也只能算是一个 UV。

3. 并发连接数

当一个网页被浏览,服务器就会和浏览器建立连接,每个连接表示一个并发。如果当前网页的页面中包含很多图片,图片并不是一个一个显示的,服务器会产生多个连接同时发送文字和图片以提高浏览速度。网页中的图片越多,那么服务器的并发连接数(Concurrent TCP Connections)就会越多。我们一般以此值作为衡量单台 Web 机器性能的参数。现在 Nginx 在网站中的应用比例非常大,可以参考 Nginx 的活动并发连接数。

4. 每秒查询率 QPS

QPS(Query Per Second)是对一个特定的查询服务器在规定时间内所处理流量多少的衡量标准,在因特网上,作为域名系统服务器的机器其性能经常用每秒查询率来衡量。对应的是 Fetches/Sec,即每秒的响应请求数,也称为最大吞吐能力。对于系统而言,QPS 数值是一个非常重要的参数,它是综合反映系统最大吞吐能力的衡量标准。它反映的不仅仅是 Web 层面的,还有缓存、数据库方面的,它反映的是系统的综合处理能力。

参考文档 <http://baike.baidu.com/view/733.htm>

参考文档 <http://www.hyokay.com/news/industry/41.html>

1.2 IDC 机房的选择及 CDN 的选型

如果自己的业务网站中含有大量的图片和视频类文件，为了加快客户端的访问速度，同时为了减缓对真正的核心机房的服务压力，并且提升用户体验，建议在前端最好采用 CDN 缓存加速方案。

CDN (Content Delivery Network)，即内容分发网络。其目的是通过在现有的 Internet 中增加一层新的网络架构，将网站的内容发布到最接近用户的网络“边缘”，使用户可以就近取得所需的内容，提高用户访问网站的响应速度。CDN 缓存加速方案一般有如下几种方式。

- ❑ 租赁 CDN：中小型网站直接购买服务就好，现在 CDN 已经进入按需付费的云计算模式了，性价比是可以准确计算的。
- ❑ 自建 CDN：这种方案的成本就有点大了，为了保证良好的缓存效果，必须在全国机房布点，还要自建智能 Bind 系统，搭建大型网站时推荐采用此种方案，专业的视频网站或图片网站一般会考虑采用此种方案。

IDC 机房的选择一般也有几种类型。

- ❑ 单电信 IDC 机房：这种类型一般业务模式比较固定，访问量也不是很大，适合新闻类网站或政务类网站。如果网站的 PV 流量持续增加的话，则建议后期采用租赁 CDN 的方式解决非电信用户访问网站速度过慢的问题。
- ❑ 双线 IDC 机房：由于国内两大网络（电信和网通）之间存在互联互通的问题，导致电信用户访问网通网站或网通用户访问电信网站速度很慢，因此就产生了双线机房、双线服务器、双线服务器托管和双线服务器租用服务。双线机房实际是一个机房有电信和网通两条线路。双线机房通过内部路由器设置，以及 BGP 自动路由的分析，可实现电信用户访问电信线路，网通用户访问网通线路，这样就可实现电信网通的快速访问。
- ❑ BGP 机房：BGP（边界网关协议）是用来连接 Internet 独立系统的路由选择协议。它是 Internet 工程任务组制定的一个加强的、完善的、可伸缩的协议。BGP4 支持 CIDR 寻址方案，该方案增加了 Internet 上的可用 IP 地址数量。BGP 是为取代最初的外部网关协议 EGP 而设计的。它也被认为是一个路径矢量协议。采用 BGP 方案来实现双线路互联或多线路互联的机房，则称为 BGP 机房。对于用户来说，选择 BGP 机房可以实现网站在各运营商线路之间互联互通，使得所有互联运营商的用户访问网站都很快，且更加稳定，不用担心全国各地因线路问题带来的访问速度快慢不一的问题，这也是传统双 IP 双线机房无法相比的优势。在条件允许的情况下，服务器的租用和托管可以尽量选择 BGP 机房，因为会带给用户最优的访问体验。

现在云计算服务也非常流行，目前首推的就是亚马逊云（AWS）和阿里云。

对于我们来说，云计算服务提供的产品能让我们的研发团队专注于产品开发本身，而不是购买、配置和维护硬件等繁杂的工作，还可以减少初始资金的投入。我们主要采用亚马逊云的 EC2/EBS/S3 服务，其实 Amazon EC2 主机提供了多种适用于不同使用案例的实例类型以供选择。实例类型由 CPU、内存、存储和网络容量组成了不同的组合，可让我们灵活地为其选择适当的资源组合。

云计算特别适合两类网站：在某些日期或某些时间段流量会激增的网站，比如竞标业务机器，用户会集中在某些时段进行竞价，因此在这些时间段使用的 Instance 数量可能是白天的几倍甚至几十倍。也就是说，此时段内瞬间可能要开启很多实例处理，处理完毕后立刻终止。EC2 Instance 是可以按照运行的小时数来进行收费的。像笔者公司的线上系统，经常运行着很多特殊业务的 Spot Instance^①，以小时计费，完成任务后立即终止。

1.3 如何根据服务器应用选购服务器

无论物理服务器是选用 IDC 托管还是 AWS EC2 云主机（以下为了简略说明，将它们统称为服务器），我们都要面临一个问题，那就是选择服务器的硬件配置，选购硬件配置时要根据服务器的应用需求而定。因为只通过一台服务器是无法满足所有需求，并解决所有的问题的。在设计网站的系统架构之前，应该从以下方面考虑如何选购服务器：

- ☐ 服务器要运行什么应用。
- ☐ 需要支持多少用户访问。
- ☐ 需要多大空间来存储数据。
- ☐ 业务有多重要。
- ☐ 服务器网卡方面的考虑。
- ☐ 安全方面的考虑。
- ☐ 机架安排是否合理化。
- ☐ 服务器的价格是否超出了预算。

1. 服务器运行什么应用

这是选购服务器时首先需要考虑的问题，通常是根据服务器的应用类型（也就是用途），来决定服务器的性能、容量和可靠性需求。下面将按照负载均衡、缓存服务器、前端服务器、应用程序服务器、数据服务器和 Hadoop 分布式计算的常见基础架构来讨论。

- ☐ 负载均衡端：除了网卡性能以外，其他方面对服务器的要求比较低，如果选用的是 LVS 负载均衡方案，那么它会直接将所有的连接要求都转给后端的 Web 应用服

^① Spot Instance：使用竞标的方式来获得便宜的 Instance，一般是在需要大量、便宜、短时间使用的需求时才会使用。

务器，因此建议选用万兆网卡。如果选用的是 HAProxy 负载均衡器，由于它的运行机制跟 LVS 不一样，流量必须双向经过 HAProxy 机器本身，因此对 CPU 的运行能力会有要求，也建议选用万兆网卡。如果选用的是 AWS EC2 机器，则推荐使用 m3.xlarge 实例类型（m3 类型提供计算、内存和网络资源的平衡，因此是很多应用程序的良好选择）。另外，AWS 官方也推出了负载均衡服务产品，即 Elastic Load Balancing，它具有 DNS 故障转移和 Auto Scalling 的功能。

- ❑ 缓存服务器：主要是 Varnish 和 redis，对 CPU 及其他方面的性能要求一般，但在内存方面的要求会尽量多些。笔者曾为了保证预算，在双核（r3.large）机器上运行了 4 个 redis 实例，AWS 官方也建议将此内存优化型实例应用于高性能数据库、分布式内存缓存、内存中分析、基因组装配和分析，以及 SAP、Microsoft SharePoint 和其他企业级应用程序的较大部署。
- ❑ 应用服务器：由于它承担了计算和功能实现的重任，因此需要为基于 Web 架构的应用程序服务器（Application Server）选择足够快的服务器，另外应用程序服务器可能需要用到大量的内存，尤其是基于 Windows 基础架构的 Ruby、Python、Java 服务器，这一类服务器至少需要使用单路至强的配置；笔者公司线上的核心业务机器选用的 AWS c3.xlarge 类型。至于可靠性问题，如果你的架构中只有一台应用服务器，这台服务器肯定要足够可靠才行，RAID 是绝对不能被忽视的选项。但如果有多台应用服务器，并设计了负载均衡机制，具有冗余功能，那就不必过于担心了。



注意 c3.xlarge EC2 主机属于计算优化型（Compute Optimized），也就是 CPU 加强型。这种类型的 CPU/内存比例比较大，适合于计算密集型业务，它包含 c1 和 c3 系列。其实例除了较旧的两个 c1 系列（c1.medium 和 c1.xlarge）是采用普通磁盘作为实例存储以外，其他的（也就是 c3 系列的）全部都以 SSD 作为实例存储，其中最高档次的 c3.8xlarge（32 核心 108 个计算单元）的网络性能明确标注为 10G bit/s，c3 系列被认为是最具性价比的类型。

- ❑ 特殊应用：除了用于 Web 架构中的应用程序之外，如果服务器还要处理流媒体视频编码、服务器虚拟化、媒体服务器，或者作为游戏服务器（逻辑、地图、聊天）运行，那么对 CPU 和内存的需求同样会比较高，至少要考虑四核以上的服务器。
- ❑ 公共服务：这里指的是邮件服务器、文件服务器、DNS 服务器、域控服务器等。通常会部署两台 DNS 服务器以互相备份，域控主服务器也会拥有一台备份服务器（专用的或非专用的），所以对于可靠性，无须过于苛刻。至于邮件服务器，至少需要具备足够的硬件可靠性和容量大小，这主要是对邮件数据负责，因为很多用户没有保存和归档邮件数据的习惯，待其重装系统后，就会习惯性地到服务器上重新下载相应的数据。至于性能问题，则应评估用户数量后再做决定。另外，考虑到它的重要

性，建议尽量选择稳定的服务器系统，比如 Linux 或 BSD 系列。

❑ 数据库服务器：数据库对服务器的要求也是最高、最重要的。无论你使用的是 MySQL、SQL Server 还是 Oracle，一般情况下，都需要有足够快的 CPU、足够大的内存、足够稳定可靠的硬件。可直接采用 DELL PowerEdge R710 或 HP 580G5，CPU 和内存方面也要尽可能最大化，如果预算充分，建议用固态硬盘做 RAID 10，因为数据库服务器对硬盘的 I/O 要求是最高的。

❑ Hadoop 分布式计算：这里建议选用密集存储实例——D2 实例，它拥有高频率 Intel Xeon E5-2676v3 (Haswell) 处理器、高达 48TB 的本地存储、具备高磁盘吞吐量，并支持 Amazon EC2 增强型联网。它适合于大规模并行处理数据仓库、MapReduce 和 Hadoop 分布式计算、分布式文件系统、网络文件系统、日志或数据处理等应用。更多关于 AWS EC2 的实例类型请参考：<https://aws.amazon.com/cn/ec2/instance-types/>。

2. 服务器需要支持多少用户访问

服务器就是用来给用户某种服务的，所以使用这些服务的用户同样是我们必须考虑的因素，可以从下面几个具体的方面进行评估：

- ❑ 有多少注册用户。
- ❑ 正常情况下有多少用户会同时在线访问。
- ❑ 每天同时在线访问的最高峰值大概是多少。

一般在项目实施之前，客户方面会针对这些问题给出一个大致的结果，但设计要尽量更充分和具体，同时，还要对未来的用户增长做一个尽可能准确的预测和规划，因为服务器可能会支持越来越多的用户，所以在进行网站或系统架构设计时要让机器能够灵活地扩展。

3. 需要多大空间来存储数据

关于这个问题需要从两个方面来考虑，一方面是有哪些类别的数据，包括：操作系统本身占用的空间，安装应用程序所需要的空间，应用程序所产生的数据、数据库、日志文件、邮件数据等，如果网站是 Web 2.0 的，还要计算每个用户的存储空间；另一方面是从时间轴上来考虑，这些数据每天都在增长，至少要为未来两三年的数据增长做个准确的预算，这就需要软件开发人员和业务人员一起来提供充分的信息了。最后将计算出来的结果乘上 1.5 左右的系数，以方便维护的时候做各种数据的备份和文件转移操作。

4. 我的业务有多重要

这需要根据自身的业务领域来考虑，下面举几个简单的例子，帮助大家了解这些服务器对可靠性、数据完整性等方面的要求：

- ❑ 如果服务器是用来运行一个 WordPress 博客的，那么，一台酷睿处理器的服务器、1GB 的内存，外加一块 160GB 的硬盘就足够了（如果是 AWS EC2 主机，可以考虑 t2.micro 实例类型）。就算服务器出现了一点硬件故障，导致几个小时甚至一两天不



能提供访问，生活也会照常继续。

- ❑ 如果服务器是用作测试平台的，那么就不会如生产环境那样对可靠性有极高的要求，所需要的可能只是做好例行的数据备份即可，若服务器宕机，只要能在当天把问题解决掉就可以了。
- ❑ 如果是一家电子商务公司的服务器，运行着电子商务网站平台，当硬件发生故障而导致宕机时，你需要对以下“危言耸听”的后果做好心理准备：投诉电话被打爆、顾客大量流失、顾客要求退款、市场推广费用打水漂、员工无事可干、公司运营陷入瘫痪状态、数据丢失。事实上，电子商务网站一般是需要 365×24 小时不间断运行和监控的，而且要有专人轮流值守，并且要有足够的备份设备，每天还要有专人负责检查。
- ❑ 如果是大型广告类或门户类网站，那么建议选择 CDN 系统。由于它具有提高网站响应速度、负载均衡、有效抵御 DDoS 攻击等特点，相对而言，每个节点都会有大量的冗余。

这里其实只是简单地讨论下业务对服务器硬件可靠性的要求。要全面解决这个问题，不能只考虑单个服务器的硬件，还需要结合系统架构的规划设计。

在回答了以上问题后，接下来就可以决定下面这些具体的选项了。

(1) 选择什么 CPU

回忆一下上面关于“服务器运行什么应用”和“需要支持多少用户访问”两个方面的考虑，这将帮助我们选择合适的 CPU。毫无疑问，CPU 的主频越高，其性能也就越高；两个 CPU 要比一个 CPU 来得更好；至强（Xeon）肯定比酷睿（Core）的性能更强。但究竟怎样的 CPU 才是最合适的呢？下面提供了一些常见情况下的建议：

- ❑ 如果业务刚刚起步，预算不是很充足，建议选择一款经典的酷睿服务器，这可以帮助你节约大量的成本。而且，以后还可以根据业务发展的情况，随时升级到更高配置的服务器。
- ❑ 如果需要在一台服务器上同时运行多种应用服务，例如基于 LNMP 架构的 Web 网站，那么一个单核至强（例如 X3330）或新一代的酷睿 i5（双核四线程）将是最佳的选择。虽然从技术的角度来说，这并不是一个好主意，但至少能节约一大笔成本。
- ❑ 如果服务器要运行 MySQL 或 Oracle 数据库，而且目前有几百个用户同时在线，未来还会不断增长，那么至少应该选择安装一个双四核服务器。
- ❑ 如果需要的是 Web 应用服务器，双四核基本就可以满足要求了。

(2) 需要多大的内存

同样，“服务器运行什么应用”和“需要支持多少用户访问”两方面的考虑也将有助于我们选择合适的内存容量。相比于 CPU，笔者认为内存（RAM）才是影响性能的最关键因素。因为在很多正在运行的服务器中，CPU 的利用率一般都在 10%~30% 之间，甚至更低。

但由于内存容量不够而导致服务器运行缓慢的案例比比皆是，如果服务器不能分配足够的内存给应用程序，那么应用程序就需要通过硬盘接口缓慢地交换读写数据了，这将导致网站慢得令人无法接受。内存的大小主要取决于服务器的用户数量，当然也和应用软件对内存的最低需求和内存管理机制有关，所以，最好由程序员或软件开发商给出最佳的内存配置建议。下面同样给出了一些常见应用环境下的内存配置建议：

- ❑ 无论是 Apache 还是 Nginx 服务器，一般情况下 Web 前端服务器都不需要配置特别高的内存，尤其是在集群架构中，4GB 的内存就已经足够了。如果用户数量持续增加，我们才会考虑使用 8GB 或更大的内存。单 Apache Web 机器，在配置了 16GB 的内存后，可以抗 6000 个并发链接数。
- ❑ 对于运行 Tomcat、Resin、WebLogic 的应用服务器，8GB 内存应该是基准配置，更准确的数字需要根据用户数量和技术架构来确定。
- ❑ 数据库服务器的内存由数据库实例的数量、表大小、索引、用户数量等来决定，一般建议配置 16GB 以上的内存，笔者公司在许多项目方案中使用了 24GB 到 48GB 的内存。
- ❑ 诸如 Postfix 和 Exchange 这样的邮件服务器对内存的要求并不高，1GB~2GB 就可以满足了。
- ❑ 还有一些特殊的服务器，需要为之配置尽可能大的内存容量，比如配置有 Varnish 和 Memcached 的缓存服务器等。
- ❑ 若是只有一台文件服务器，1GB 的内存可能就足够了。

事实上，由于内存技术在不断进化，价格也在不断降低，因此才得以近乎奢侈地讨论 4GB、8GB、16GB 这些曾经不可想象的内存容量。然而，除了花钱购买内存来满足应用程序的“贪婪”之外，系统优化和数据库优化仍然是我们需要重视的问题。

(3) 需要怎样的硬盘存储系统

硬盘存储系统的选择和配置是整个服务器系统里最复杂的一部分，需要考虑硬盘的数量、容量、接口类型、转速、缓存大小，以及是否需要 RAID 卡、RAID 卡的型号和 RAID 级别等问题。甚至在一些高可靠性高性能的应用环境中，还需要考虑使用怎样的外部存储系统 (SAN、NAS 或 DAS)。下面将服务器的硬盘 RAID 卡的特点归纳一下：

- ❑ 如果是用作缓存服务器，比如 Varnish 或 redis，则可以考虑用 RAID 0。
- ❑ 如果是跑 Nginx+FastCGI 或 Nginx 等应用，则可以考虑用 RAID 1。
- ❑ 如果是内网开发服务器或存放重要代码的服务器，则可以考虑用 RAID 5。
- ❑ 如果是跑 MySQL 或 Oracle 等数据库应用，则可以考虑用固态硬盘做 RAID 5 或 RAID 10。

5. 网卡性能方面的考虑

如果基础架构是多服务器环境，而且服务器之间有大量的数据交换，那么建议为每台服务器配置两个或更多的网卡，一个用来对外提供服务，另一个用来做内部数据交换。由

于现在项目外端都置于防火墙内，所以许多时候单网卡就足够了；而比如 LVS+Keepalived 这种只用公网地址的 Linux 集群架构，有时可能只需要一块网卡即可。建议大家选用万兆网卡。另外，建议交换机至少也要选择千兆网卡级别的。

如果采用的是 AWS EC2 云主机环境，单纯以 EC2 作为 LVS 或 HAProxy 意义不大。如果经常使用 AWS EC2 机器，应该会注意到 AWS 将机器的网卡性能分成了 3 个级别，即 Low、Moderate、High，那么这 3 个级别分别是什么情况呢？虽然 AWS 没有带宽限制，但是由于多虚拟机共享 HOST 物理机的网络性能和 I/O 性能，单个虚拟机的网络性能也不是特别好度量，不过大概情况是这样的：Low 级别的是 20M bit/s，Moderate 级别的是 40M bit/s，High 级别的能达到 80M bit/s~100M bit/s。从上面的分析情况可以得知，单台 AWS EC2 主机作为网站的负载均衡入口，容易成为网站的瓶颈。这个时候可以考虑使用 AWS 提供的 Elastic Load Balancing 产品，它可以在云中的多个 Amazon EC2 实例间自动分配应用程序的访问流量（大家注意到没，相当于它将网站的流量分担到了多台机器上）。它可以实现更高水平的应用程序容错性能，从而无缝地提供分配应用程序流量所需的负载均衡容量。除了提供负载均衡常见的功能之外，它还具有 Auto Scalling 功能，关于 Auto Scalling 的详细介绍，可参见 AWS 官方文档。

6. 服务器安全方面的考虑

由于目前国内的 DDoS 攻击还是比较普遍的，因此建议给每个项目方案和自己的电子商务网站配备硬件防火墙，比如 Juniper、Cisco 等硬件防火墙。当然了，这个问题也是网站后期运营维护需要考虑的，这里只是想让大家有个概念性的认识。此外，建议租赁 CDN 服务，这样万一不幸遭遇恶意的 DDoS 流量攻击，CDN 还能帮助抵挡部分恶意流量，核心机房的业务不至于在很短的时间内就会崩溃。

7. 根据机架数合理安排服务器的数量

这个问题应该在项目实施前就处理好了的，选择服务器时应该明确服务器的规格，到底是 1U、2U 还是 4U 的，到底有多少台服务器和交换机，应该如何安排，毕竟机柜只有 42U 的容量。在小项目中这个问题可能无关紧要，但在大型项目的实施过程中，这个问题就很突出了。我们应该根据现有或额定的机架数目确定到底应该选择多少台服务器和交换机。

8. 成本考虑：服务器的价格问题

无论是在公司采购时，还是在项目实施过程中，成本都是非常重要的问题。笔者的方案经常被退回，理由就是超出预算。尤其对于一些小项目，预算更吃紧。之前笔者经常面对的客户需求是为证券类资讯网站设计方案，只要求周一至周五的上午九点至下午三点网站不出问题即可，并不需要做复杂的负载均衡高可用。所以面对这种需求，笔者会将单 Nginx 或 HAProxy 设计成负载均衡，后面接两台 Web 应用服务器作为简单集群架构。如果是做中大型电子商务网站，那么在服务器成本上的控制就尤其重要了。事实上，我们经

常面对的问题是，客户给出的成本预算有限，而实际应用又需要比较多的服务器，这时候，就不得不另外设计一套最小化成本预算方案来折中处理。

以上8个方面是我们在采购服务器时应该要注意的因素，在选择服务器的组件时要有所偏重，然后根据系统或网站架构来决定服务器的数量，尽量做到服务器资源利用的最大化。在控制方案成本的同时，要做到最优的性价比。

1.4 CentOS 6.4 x86_64 最小化安装后的优化

购买了服务器以后要做的第一件事就是安装操作系统了，这里推荐安装 CentOS 6.4 x86_64，安装系统时要选择最小化安装（不需要图形），在使用服务器时要记住一个原则，系统安装的应用程序包越少，服务器就会越稳定。至于服务器单机性能调优，应本着稳定安全的原则，尽量不要改动系统原有的配置（CentOS 系统自身的文件和内存机制就很优秀），以下配置优化部分也适合 Amazon Linux 系统，大家可以对比参考。

1.4.1 系统的基础优化

1. 更新 yum 官方源

CentOS 6.4 系统自带的更新源速度比较慢，想必各位都有所感受，国内的速度慢得让人受不了。为了让 CentOS 6.4 系统使用速度更快的 yum 更新源，一般做运维的都会选择更换源，笔者一般会选择网易的更新源，详细步骤如下所示。

1) 下载 repo 文件，命令如下：

```
wget http://mirrors.163.com/.help/CentOS6-Base-163.repo
```

2) 备份并替换系统的 repo 文件，命令如下：

```
cd /etc/yum.repos.d/
mv CentOS-Base.repo CentOS-Base.repo.bak
mv CentOS6-Base-163.repo CentOS-Base.repo
```

3) 执行 yum 源更新，命令如下：

```
yum clean all #清除yum缓存
yum makecache #重建缓存
yum update #升级Linux系统
```

增加 epel 源，详细步骤如下所示。

1) 下载 rpm 文件并进行安装，命令如下：

```
wget http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
rpm -ivh epel-release-6-8.noarch.rpm
```

2) 安装 yum-priorities 源优先级工具，命令如下：

```
yum install yum-priorities
```

yum-priorities 源优先级工具是 yum-plugin-priorities 插件，用于给 yum 源划分优先级。比如说系统存在官方源、epel、puppetlabs3 个 yum 源，这 3 个 yum 源中可能含有相同的软件，yum 管理器会应用该工具来分辨安装软件时采用哪个 yum 源的软件。

如果说，设置官方的 yum 源优先级最高，epel yum 源第二，puppetlabs 第三（用 1 到 99 来表示，1 最高），那么在安装程序的时候，就会先寻找官方的 yum 源。如果该源里面有所要的程序，那就停止寻找，直接安装找到的；如果没有找到，则依次寻找 epel 和 puppetlabs 的源。如果说 3 个 yum 源都含有同一个软件，那就安装优先级最高的官方 yum 源的。添加优先级的过程比较简单，只需要编辑对应的 repo 文件，在文件最后添加如下内容即可：

priority=对应优先级数字

注意，要想开启 yum 源的优先级功能，就要先确保 priorities.conf 文件里面有如下内容，需要先打开此文件，打开文件的命令如下：

```
vim /etc/yum/pluginconf.d/priorities.conf
```

确保文件里面包含如下内容：

```
[main]
enabled=1
```

2. 关闭不需要的服务

众所周知，服务越少，系统占用的资源就会越少，所以应当关闭不需要的服务。建议把不需要的服务关闭掉，这样做的好处是减少内存和 CPU 资源占用。首先可以看下系统中存在着哪些已经启动的服务，查看命令如下：

```
ntsysv
```

下面列出的是需要启动的服务，未列出的服务一律关闭。

- ☐ crond：自动计划任务。
- ☐ network：Linux 系统的网络服务，很重要，若不开启此服务的话，服务器就不能联网。
- ☐ sshd：OpenSSH 服务器守护进程。
- ☐ rsyslog：Linux 的日志系统服务（CentOS 5.8 下此服务名称为 syslog），必须要启动。

3. 关闭不需要的 TTY

可用 vim 编辑器打开 vim /etc/init/start-ttys.conf 文件，文件内容如下所示：

```
start on stopped rc RUNLEVEL=[2345]
env ACTIVE_CONSOLES=/dev/tty[1-6]
env X_TTY=/dev/tty1
task
script
```

```

. /etc/sysconfig/init
for tty in $(echo $ACTIVE_CONSOLES) ; do
    [ "$$RUNLEVEL" = "5" -a "$tty" = "$X_TTY" ] && continue
    initctl start tty TTY=$tty
done
end script

```

这段代码使 init 打开了 6 个控制台，可分别用 ALT+F1 到 ALT+F6 进行访问。此 6 个控制台默认都驻留在内存中，用 ps aux 命令即可看到，命令如下：

```
ps aux | grep tty | grpe -v grep
```

命令显示结果如下所示：

```

root      1011  0.0  0.1  4060   288 tty1      Ss+  Nov25   0:00 /sbin/mingetty /dev/tty1
root      1013  0.0  0.1  4060   292 tty2      Ss+  Nov25   0:00 /sbin/mingetty /dev/tty2
root      1015  0.0  0.1  4060   292 tty3      Ss+  Nov25   0:00 /sbin/mingetty /dev/tty3
root      1017  0.0  0.1  4060   288 tty4      Ss+  Nov25   0:00 /sbin/mingetty /dev/tty4
root      1019  0.0  0.1  4060   288 tty5      Ss+  Nov25   0:00 /sbin/mingetty /dev/tty5
root      1021  0.0  0.1  4060   292 tty6      Ss+  Nov25   0:00 /sbin/mingetty /dev/tty6
root      7467  0.0  0.1  4072   340 hvc0      Ss+  Nov29   0:00 /sbin/agetty /dev/hvc0
38400 vt100-nav

```

事实上没有必要使用这么多，那如何关闭不需要的进程呢？

通常保留两个控制台就可以了，打开 /etc/init/start-ttys.conf 文件，注意以下代码内容：

```
env ACTIVE_CONSOLES=/dev/tty[1-6]
```

将 [1-6] 修改为 [1-2]，然后再打开 /etc/sysconfig/init 文件，注意以下代码内容：

```
ACTIVE_CONSOLES=/dev/tty[1-6]
```

将 [1-6] 修改为 [1-2]，然后重启机器即可。

4. 对 TCP/IP 网络参数进行调整

调整 TCP/IP 网络参数，可以加强对抗 SYN Flood 的能力，命令如下：

```
echo 'net.ipv4.tcp_syncookies = 1' >> /etc/sysctl.conf
sysctl -p
```

5. 修改 SHELL 命令的 history 记录个数

用 vim 编辑器打开 /etc/profile 文件，关注 HISTSIZE=1000：

```
vi /etc/profile
```

在找到 HISTSIZE=1000 后，将其改为 HISTSIZE=100（这条可根据实际工作环境而定）。

不需要重启系统也可让其生效，命令如下：

```
source /etc/profile
```


6. 定时校正服务器的时间

我们可以定时校正服务器的时间，命令如下：

```
yum install ntp
crontab -e
```

加入一行：

```
* /5 * * * * /usr/sbin/ntpdate ntp.api.bz
```

ntp.api.bz 是一组 NTP 服务器集群，之前是 6 台服务器，位于上海电信；现在是 3 台服务器，分散于上海和浙江电信，可以用 dig 命令查看：

```
dig ntp.api.bz
```

命令显示结果如下所示：

```
; <<>> DiG 9.8.2rc1-RedHat-9.8.2-0.37.rc1.el6_7.5 <<>> ntp.api.bz
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 48560
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
ntp.api.bz.                IN      A
;; ANSWER SECTION:
ntp.api.bz.                600     IN      A      61.153.197.226
ntp.api.bz.                600     IN      A      218.75.4.130
ntp.api.bz.                600     IN      A      114.80.81.1
;; Query time: 10 msec
;; SERVER: 202.103.24.68#53(202.103.24.68)
;; WHEN: Thu Dec 24 06:58:36 2015
;; MSG SIZE rcvd: 76
```

7. 停止 IPv6 网络服务

在 CentOS 6.4 默认的状态下，IPv6 是被启用的，可用如下命令查看：

```
lsmod | grep ipv6
```

命令显示结果如下：

```
nf_conntrack_ipv6          8748  2
nf_defrag_ipv6             11182  1 nf_conntrack_ipv6
nf_conntrack               79357  2 nf_conntrack_ipv6,xt_state
ipv6                       321422  23 ip6t_REJECT,nf_conntrack_ipv6,nf_defrag_ipv6
```

有些网络 and 应用程序还不支持 IPv6，因此，禁用 IPv6 可以说是一个非常好的选择，以加强系统的安全性，并提高系统的整体性能。不过，首先要确认一下 IPv6 是不是处于被启动的状态，命令如下：

```
ifconfig -a ← 列出全部网络接口信息
```

命令显示结果如下所示：

```

eth0      Link encap:Ethernet  HWaddr 00:16:3E:7F:67:C3
          inet addr:192.168.1.207  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::216:3eff:fe7f:67c3/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:405835 errors:0 dropped:0 overruns:0 frame:0
          TX packets:197486 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:327950786 (312.7 MiB)  TX bytes:17186162 (16.3 MiB)
          Interrupt:24

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

```

然后修改相应的配置文件，停止 IPv6，命令如下：

```

echo "install ipv6 /bin/true" > /etc/modprobe.d/disable-ipv6.conf
#每当系统需要加载IPv6模块时，强制执行/bin/true来代替实际加载的模块
echo "IPV6INIT=no" >> /etc/sysconfig/network-scripts/ifcfg-eth0
#禁用基于IPv6网络，使之不会被触发启动

```

8. 调整 Linux 的最大文件打开数

要调整一下 Linux 的最大文件打开数，否则运行 Squid 服务的机器在高负载时执行性能将会很差；另外，在 Linux 下部署应用时，有时候会遇上“Too many open files”这样的问题，这个值也会影响服务器的最大并发数。其实 Linux 是有文件句柄限制的，但默认值不是很高，一般是 1024，生产服务器很容易就会达到这个值，所以需要改动此值。

下面打开 /etc/security/limit.conf 命令，在最后一行添加如下命令：

```

* soft nofile 65535
* hard nofile 65535

```

正解的做法应该是除了以上步骤之外，还要在系统的 /etc/rc.local 文件里添加如下内容：

```
ulimit -SHn 65535
```

另外，ulimit -n 命令并不能真正看到文件的最大文件打开数，可用如下脚本查看：

```

#!/bin/bash
for pid in `ps aux |grep nginx |grep -v grep|awk '{print $2}'`
do
cat /proc/${pid}/limits |grep 'Max open files'
done

```

在线上环境找一台 cms 业务机器，执行此脚本，显示结果如下所示：

```

Max open files      65535      65535      files

```

```

Max open files      65535      65535      files
Max open files      65535      65535      files
Max open files      65535      65535      files
Max open files      65535      65535      files
Max open files      65535      65535      files
Max open files      65535      65535      files
Max open files      65535      65535      files
Max open files      65535      65535      files
Max open files      65535      65535      files

```

9. 启动网卡

在配置 CentOS 6.4 的网卡 IP 地址时，容易忽略的一项是 Linux 在启动时未启动网卡，其后果很明显，那就是该 Linux 机器永远也没有 IP 地址。下面是一台线上环境服务器网卡 `/etc/sysconfig/network-scripts/ifcfg-eth0` 文件的配置内容：

```

DEVICE=eth0
BOOTPROTO=static
HWADDR=00:14:22:1B:71:20
IPV6INIT=no
IPV6_AUTOCONF=yes
ONBOOT=yes    -->此项一定要记得更改为yes,它会在系统引导时就启动网卡设备
NETMASK=255.255.255.192
IPADDR=203.93.236.146
GATEWAY=203.93.236.129
TYPE=Ethernet
PEERDNS=yes   -->允许用从DHCP处获得的DNS覆盖本地的DNS
USERCTL=no    -->不允许普通用户修改网卡

```

10. 关闭写磁盘 I/O 功能

Linux 文件默认有 3 个时间，分别如下所示。

- ☐ `atime`：对此文件的访问时间。
- ☐ `ctime`：此文件 inode 发生变化的时间。
- ☐ `mtime`：此文件的修改时间。

如果有多个小文件（比如 Web 服务器的页面上有多个小图片），通常是没有必要记录文件的访问时间的，这样就可以减少写磁盘的 I/O，可这要如何配置呢？

首先，修改文件系统的配置文件 `/etc/fstab`，然后，在包含大量小文件的分区中使用 `noatime` 和 `nodiratime` 这两个命令。例如：

```
/dev/sda5 /data/pics ext3 noatime,nodiratime 0 0
```

这样文件被访问时就不会再产生写磁盘的 I/O 了。

11. 修改 SSH 登录配置

SSH 服务配置优化，请保持机器中至少包含一个具有 `sudo` 权限的用户，下面的配置会禁止 `root` 远程登录，代码如下所示：

```
sed -i 's@#PermitRootLogin yes@PermitRootLogin no@' /etc/ssh/sshd_config #禁止root远程登录
sed -i 's@#PermitEmptyPasswords no@PermitEmptyPasswords no@' /etc/ssh/sshd_config
#禁止空密码登录
sed -i 's@#UseDNS yes@UseDNS no@' /etc/ssh/sshd_config /etc/ssh/sshd_config #关闭
SSH反向查询，以加快SSH的访问速度
```

12. 增加具有 sudo 权限的用户

添加用户的步骤和过程比较简单（这里略过），由于系统已经禁止了 root 远程登录，因此需要一个具有 sudo 权限的 admin 用户，权限跟 root 相当，这里用 vim 命令，在打开的 /etc/sudoers 文件内容里添加如下内容：

```
## Allow root to run any commands anywhere
root    ALL=(ALL)        ALL
```

然后添加如下内容：

```
admin    ALL=(ALL)        ALL
```

如果在进行 sudo 切换时不想输入密码，可以做如下更改：

```
admin    ALL=(ALL) NOPASSWD:ALL
```

1.4.2 优化 Linux 下的内核 TCP 参数以提高系统性能

内核的优化跟服务器的优化一样，应本着稳定安全的原则。下面以 Squid 服务器为例来说明，待客户端与服务器端建立 TCP/IP 连接后就会关闭 Socket，服务器端连接的端口状态也就变为 TIME_WAIT 了。那是不是所有执行主动关闭的 Socket 都会进入 TIME_WAIT 状态呢？有没有什么情况可使主动关闭的 Socket 直接进入 CLOSED 状态呢？答案是主动关闭的一方在发送最后一个 ACK 后就会进入 TIME_WAIT 状态，并停留 2MSL（报文最大生存）时间，这是 TCP/IP 必不可少的，也就是说这一点是“解决”不了的。

TCP/IP 设计者如此设计，主要原因有两个：

- ❑ 防止上一次连接中的包迷路后重新出现，影响新的连接（经过 2MSL 时间后，上一次连接中所有重复的包都会消失）。
- ❑ 为了可靠地关闭 TCP 连接。主动关闭方发送的最后一个 ACK (FIN) 有可能会丢失，如果丢失，被动方会重新发送 FIN，这时如果主动方处于 CLOSED 状态，就会响应 RST 而不是 ACK。所以主动方要处于 TIME_WAIT 状态，而不能是 CLOSED 状态。另外，TIME_WAIT 并不会占用很大的资源，除非受到攻击。

在 Squid 服务器中可输入如下命令查看当前连接统计数：

```
netstat -n | awk '/^tcp/ {++S[$NF]} END{for(a in S) print a, S[a}]'
```

命令显示结果如下所示：

```
LAST_ACK 14
SYN_RECV 348
```



```
ESTABLISHED 70
FIN_WAIT1 229
FIN_WAIT2 30
CLOSING 33
TIME_WAIT 18122
```

命令中的含义分别如下。

- ❑ CLOSED: 无活动的或正在进行的连接。
- ❑ LISTEN: 服务器正在等待进入呼叫。
- ❑ SYN_RECV: 一个连接请求已经到达, 等待确认。
- ❑ SYN_SENT: 应用已经开始, 打开一个连接。
- ❑ ESTABLISHED: 正常数据传输状态。
- ❑ FIN_WAIT1: 应用说它已经完成。
- ❑ FIN_WAIT2: 另一边已同意释放。
- ❑ ITMED_WAIT: 等待所有分组死掉。
- ❑ CLOSING: 两边尝试同时关闭。
- ❑ TIME_WAIT: 另一边已初始化一个释放。
- ❑ LAST_ACK: 等待所有分组死掉。

也就是说, 这条命令可以把当前系统的网络连接状态分类汇总。

在 Linux 下高并发的 Squid 服务器中, TCP TIME_WAIT 套接字的数量经常可达到两三万, 服务器很容易就会被拖死。不过, 可以通过修改 Linux 内核参数来减少 Squid 服务器的 TIME_WAIT 套接字数量, 命令如下:

```
vim /etc/sysctl.conf
```

然后, 增加以下参数:

```
net.ipv4.tcp_fin_timeout = 30
net.ipv4.tcp_keepalive_time = 1200
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.ip_local_port_range = 10000 65000
net.ipv4.tcp_max_syn_backlog = 8192
net.ipv4.tcp_max_tw_buckets = 5000
```

以下将简单说明上面各个参数的含义:

- ❑ net.ipv4.tcp_syncookies=1 表示开启 SYN Cookies。当出现 SYN 等待队列溢出时, 启用 Cookie 来处理, 可防范少量的 SYN 攻击。该参数默认为 0, 表示关闭。
- ❑ net.ipv4.tcp_tw_reuse=1 表示开启重用, 即允许将 TIME-WAIT 套接字重新用于新的 TCP 连接。该参数默认为 0, 表示关闭。
- ❑ net.ipv4.tcp_tw_recycle=1 表示开启 TCP 连接中 TIME-WAIT 套接字的快速回收, 该参数默认为 0, 表示关闭。

- ❑ `net.ipv4.tcp_fin_timeout=30` 表示如果套接字由本端要求关闭, 那么这个参数将决定它保持在 FIN-WAIT-2 状态的时间。
- ❑ `net.ipv4.tcp_keepalive_time=1200` 表示当 Keepalived 启用时, TCP 发送 Keepalived 消息的频率改为 20 分钟, 默认值是 2 小时。
- ❑ `net.ipv4.ip_local_port_range=10 000 65 000` 表示 CentOS 系统向外连接的端口范围。其默认值很小, 这里改为 10 000 到 65 000。建议不要将这里的最低值设得太低, 否则可能会占用正常的端口。
- ❑ `net.ipv4.tcp_max_syn_backlog=8192` 表示 SYN 队列的长度, 默认值为 1024, 此处加大队列长度为 8192, 可以容纳更多等待连接的网络连接数。
- ❑ `net.ipv4.tcp_max_tw_buckets=5000` 表示系统同时保持 TIME_WAIT 套接字的最大数量, 如果超过这个数字, TIME_WAIT 套接字将立刻被清除并打印警告信息, 默认值为 180 000, 此处改为 5000。对于 Apache、Nginx 等服务器, 前面介绍的几个参数已经可以很好地减少 TIME_WAIT 套接字的数量, 但是对于 Squid 来说, 效果却不大, 有了此参数就可以控制 TIME_WAIT 套接字的最大数量, 避免 Squid 服务器被大量的 TIME_WAIT 套接字拖死。

执行以下命令使内核配置立马生效:

```
/sbin/sysctl -p
```

如果是用于 Apache 或 Nginx 等 Web 服务器, 则只需要更改以下几项即可:

```
net.ipv4.tcp_syncookies=1
net.ipv4.tcp_tw_reuse=1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.ip_local_port_range = 10000 65000
```

执行以下命令使内核配置立马生效:

```
/sbin/sysctl -p
```

如果是 Postfix 邮件服务器, 则建议内核优化方案如下:

```
net.ipv4.tcp_fin_timeout = 30
net.ipv4.tcp_keepalive_time = 300
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.ip_local_port_range = 10000 65000
kernel.shmmax = 134217728
```

执行以下命令使内核配置立马生效:

```
/sbin/sysctl -p
```

当然这些都只是最基本的更改, 大家还可以根据自己的需求来更改内核的设置, 比如我们的线上机器在高并发的情况下, 经常会出现 “TCP: too many orphaned sockets” 的报错。内核调优尽量也要本着服务器稳定的最高原则。如果服务器不稳定的话, 一切工作和努力就都会白费。

如果以上优化仍无法满足工作要求，则有可能需要定制你的服务器内核或升级服务器硬件。

1.4.3 CentOS 6.4 x86_64 系统最小化优化脚本

CentOS 6.4 x86_64 系统最小化优化脚本，脚本内容如下所示（请注意下面的代码中有中文注释内容，如果是放在线上运行时则要注意）：

```
#!/bin/bash
#系统基础升级
wget http://mirrors.163.com/.help/CentOS6-Base-163.repo
cd /etc/yum.repos.d/
mv CentOS-Base.repo CentOS-Base.repo.bak
mv CentOS6-Base-163.repo CentOS-Base.repo
yum clean all #清除yum缓存
yum makecache #重建缓存
yum update #升级Linux系统
#添加epel外部yum扩展源
cd /usr/local/src
wget http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
rpm -ivh epel-release-6-8.noarch.rpm
#安装gcc基础库文件及sysstat工具
yum -y install gcc gcc-c++ vim-enhanced unzip unrar sysstat
#配置ntpd自动对时
yum -y install ntp
echo "01 01 * * * /usr/sbin/ntpdate ntp.api.bz    >> /dev/null 2>&1" >> /etc/crontab
ntpdate ntp.api.bz
service crond restart
#配置文件的ulimit值
ulimit -SHn 65534
echo "ulimit -SHn 65534" >> /etc/rc.local
cat >> /etc/security/limits.conf << EOF
*                                soft          nofile          65534
*                                hard          nofile          65534
EOF

#基础系统内核优化
cat >> /etc/sysctl.conf << EOF
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_syn_retries = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_fin_timeout = 1
net.ipv4.tcp_keepalive_time = 1200
net.ipv4.ip_local_port_range = 10000 65535
net.ipv4.tcp_max_syn_backlog = 16384
net.ipv4.tcp_max_tw_buckets = 36000
net.ipv4.route.gc_timeout = 100
net.ipv4.tcp_syn_retries = 1
net.ipv4.tcp_synack_retries = 1
net.core.somaxconn = 16384
```

```

net.core.netdev_max_backlog = 16384
net.ipv4.tcp_max_orphans = 16384
EOF
/sbin/sysctl -p

#禁用control-alt-delete组合键以防止误操作
sed -i 's@ca::ctrlaltdel:/sbin/shutdown -t3 -r now@ca::ctrlaltdel:/sbin/shutdown
-t3 -r now@' /etc/inittab
#关闭SELinux
sed -i 's@SELINUX=enforcing@SELINUX=disabled@' /etc/selinux/config
#关闭iptables
service iptables stop
chkconfig iptables off
#ssh服务配置优化,请保持机器中至少存在一个具有sudo权限的用户,下面的配置会禁止root远程登录
sed -i 's@#PermitRootLogin yes@PermitRootLogin no@' /etc/ssh/sshd_config
#禁止空密码登录
sed -i 's@#PermitEmptyPasswords no@PermitEmptyPasswords no@' /etc/ssh/sshd_config
#禁止SSH反向解析
sed -i 's@#UseDNS yes@UseDNS no@' /etc/ssh/sshd_config /etc/ssh/sshd_config
service sshd restart
#禁用IPv6地址
echo "install ipv6 /bin/true" > /etc/modprobe.d/disable-ipv6.conf
#每当系统需要加载IPv6模块时,强制执行/bin/true来代替实际加载的模块
echo "IPV6INIT=no" >> /etc/sysconfig/network-scripts/ifcfg-eth0
#禁用基于IPv6网络,使之不会被触发启动
chkconfig ip6tables off
#vim基础语法优化
cat >> /root/.vimrc << EOF
set number
set ruler
set nohlsearch
set shiftwidth=2
set tabstop=4
set expandtab
set cindent
set autoindent
set mouse=v
syntax on
EOF
#精简开机自启动服务,安装最小化服务的机器初始可以只保留crond|network|rsyslog|sshd这4个服务
for i in `chkconfig --list|grep 3:on|awk '{print $1}'`;do chkconfig --level 3 $i off;done
for CURSRV in crond rsyslog sshd network;do chkconfig --level 3 $CURSRV on;done
#重启服务器
reboot

```

1.4.4 Linux 下 CPU 使用率与机器负载的关系与区别

笔者的线上竞标业务机器,在业务最繁忙的一段周期内,发现 Nginx 单机并发活动的连接数超过了 2.6 万,机器负载(基本上不到 4, Nagios 监控系统并没有发送报警邮件和短信)和 Nginx+Lua 服务都是正常的,网卡流量并没有打满,但流量就是怎么也打不进

去。经过深入观察，发现这段时期内每台机器的 CPU 利用率都已经很高了，基本都维持在 99%~100% 左右，这种情况应该是 CPU 资源耗尽了，导致不能再继续提供服务，所以这里有必要研究下 CPU 负载和 CPU 利用率这两个概念的关系与区别。

CPU 负载和 CPU 利用率虽然是两个不同的概念，但它们的信息可以显示在同一个 top 命令中。CPU 利用率显示的是程序在运行期间实时占用的 CPU 百分比，而 CPU 负载显示的则是一段时间内正在使用 and 等待使用 CPU 的平均任务数。CPU 利用率高，并不意味着负载就一定大。

网上有篇文章提供了一个有趣的比喻，即通过打电话来说明两者的区别，下面笔者按照自己的理解来阐述一下。

某公用电话亭，有一个人在打电话，四个人在等待，每人限定使用电话一分钟，若有人一分钟之内没有打完电话，则只能挂掉电话去排队，等待下一轮。电话在这里就相当于 CPU，而正在或等待打电话的人就相当于任务数。

在电话亭的使用过程中，肯定会有人打完电话走掉，有人没有打完电话而选择重新排队，更会有新增的人在这儿排队，这个人数的变化就相当于任务数的增减。为了统计平均负载情况，我们 5 秒钟统计一次人数，并在第 1、5、15 分钟的时候对统计情况取平均值，从而形成第 1、5、15 分钟的平均负载。有的人拿起电话就打，一直打完一分钟；而有的人可能前三十秒在找电话号码，或者在犹豫要不要打，后三十秒才真正在打电话。如果把电话看作 CPU，人数看作任务，可以说前一个人（任务）的 CPU 利用率高，后一个人（任务）的 CPU 利用率低。

当然，CPU 并不会在前三十秒工作，后三十秒歇着，只是说，有的程序涉及大量的计算，所以 CPU 利用率就高，而有的程序涉及计算的部分很少，CPU 利用率自然就低。但无论 CPU 的利用率是高还是低，都跟后面有多少任务在排队没有必然关系。

CPU 负载为多少才算比较理想的呢？对于这个问题，业界一直存在争议，各有各的说法，个人比较赞同 CPU 负载小于等于 0.5 算是一种理想的状态。

不管某个 CPU 的性能有多好，1 秒钟能处理多少个任务，我们都可以认为它无关紧要，虽然事实并非如此。在评估 CPU 负载时，我们只以 5 秒钟为单位来统计任务队列长度。如果每隔 5 秒钟统计的时候，发现任务队列的长度都是 1，那么 CPU 负载就为 1。假如只有一个单核的 CPU，负载一直为 1，那就意味着没有任务在排队，这说明目前机器系统负载还不错。还是以上面提到的竞标业务机器为例，都是四核机器，每个内核的负载为 1 的话，则总负载为 4。也就是说，如果那些竞标服务器的 CPU 负载长期保持在 4 左右，还是可以接受的。

CPU 使用率达到多少才算比较理想？

建议统计 %user+%system 的值（后面的章节有相关的 Shell 脚本），如果长期大于 85% 的话，就可以认为系统的 CPU 负载过重，这个时候就可以考虑添加物理 CPU 或增添业务集群机器了。

1.5 MySQL 数据库的优化

网站上线初期，由于业务和名气的原因，访问量一般会处于一个比较低水平。但随着业务量的扩大，网站宣传力度的增加，网站的 PV 和 UV 日渐增多，后端的 MySQL 数据库压力也越来越大，网站的查询功能或订单系统会越来越慢，那么客户的用户体验是相当糟糕的。那么究竟应该如何对 MySQL 数据库进行优化呢？下面就从 MySQL 服务器对硬件的选择、配置文件的优化等方面来说明这个问题。

1.5.1 服务器物理硬件的优化

在对 MySQL 服务器进行硬件挑选时，应该从下面几个方面着重对 MySQL 服务器的硬件配置进行优化，也就是说将项目中的资金着重投入到如下几处：

- ❑ 磁盘寻道能力（磁盘 I/O）。笔者公司现在用的都是 SAS15000 转的硬盘，用 6 块这样的硬盘做 RAID 10。MySQL 数据库每一秒钟都在进行大量、复杂的查询操作，对磁盘的读写量可想而知，所以，通常认为磁盘 I/O 是制约 MySQL 性能的最大因素之一。对于日均访问量在 1000 万 PV 以上的 Discuz 论坛，如果磁盘 I/O 性能不好，造成的直接后果就是 MySQL 的性能会非常低下！解决这一制约因素可以考虑的解决方案是：使用 RAID 10 磁盘阵列，注意不要使用 RAID 5 磁盘阵列。MySQL 在 RAID5 磁盘阵列上的效率不会像你期待的那样快，如果资金条件允许，可以选择固态硬盘 SSD 来代替 SAS 硬盘做 RAID 10。
- ❑ CPU 对于 MySQL 的影响也是不容忽视的，建议选择运算能力强悍的 CPU；推荐使用 DELL PowerEdge R710，英特尔双至强 E5504（四核心高性能 CPU），该产品的卖点就是强大的虚拟化和数据处理能力。当然了，如果资金允许，可以考虑下更高级别的 DELL PowerEdge R910。
- ❑ 对于一台使用 MySQL 的数据库服务器而言，建议服务器的内存不要低于 16GB，不过对于现在的服务器而言内存的大小是一个可以忽略的问题，如果是高端服务器，基本上内存都超过了 32GB，笔者公司的数据库服务器都是 32GB DDR3 的内存。
- ❑ 强烈建议 MySQL 数据库服务器的系统为 64 操作系统（无论使用的是 Windows 系统还是 Linux 系统，如果没有特殊原因，建议 MySQL 数据库还是运行在 64 位的操作系统上），32 位的系统存在非常多的制约。

1.5.2 利用 tuning-primer 脚本来调优 MySQL 数据库

MySQL 在线上稳定运行一段时间后，就可以调用 MySQL 调优脚本 tuning-primer.sh 来检查参数的设置是否合理，该脚本的下载地址为：

<http://www.day32.com/MySQL/tuning-primer.sh>。

该脚本使用“SHOW STATUS LIKE…”和“SHOW VARIABLES LIKE…”命令获得 MySQL 的相关变量和运行状态。然后根据推荐的调优参数对当前的 MySQL 数据库进行测试。最后根据不同颜色的标识来提醒用户需要注意的各个参数设置。

当前版本会处理如下这些推荐的参数：

- ☐ Slow Query Log (慢查询日志)
- ☐ Max Connections (最大连接数)
- ☐ Worker Threads (工作线程)
- ☐ Key Buffer (Key 缓冲)
- ☐ Query Cache (查询缓存)
- ☐ Sort Buffer (排序缓存)
- ☐ Joins (连接)
- ☐ Temp Tables (临时表)
- ☐ Table (Open & Definition) Cache (表缓存)
- ☐ Table Locking (表锁定)
- ☐ Table Scans (read_buffer)(表扫描, 读缓冲)
- ☐ InnoDB Status (InnoDB 状态)

笔者之前所在公司的主营业务是 CPA 电子广告平台, 公司规模比较小, 所以没有配备专业的 MySQL DBA, 线上的 MySQL 数据库 (四核 CPU) 服务器问题比较多, 用 tuning-primer.sh 脚本扫描后发现如下问题:

- ☐ MySQL 数据库有时连接非常慢, 严重时会被拖死。

通过 show full processlist 命令可以发现大量的“unauthenticated user”连接, 数据库肯定每次都要响应, 所以速度越来越慢, 解决方法其实很简单: 在 mysql.cnf 里添加 skip-name-resolve, 即不启用 DNS 反向解析。

发生这种情况的原因其实也很简单, MySQL 的认证实际上是 user+host 的形式 (也就是说 user 可以相同), 所以 MySQL 在处理新连接时会试着去解析客户端连接的 IP, 启用参数 skip-name-resolve 后 MySQL 授权的时候就只能使用纯 IP 的形式了。

- ☐ 数据库在繁忙期间负载很大, 长期达到了 13, 远远超过了系统平均负载 4, 这个肯定是不正常的。

通过脚本扫描, 发现没有新建 thread_cache_size, 所以加上了 thread_cache_size=256, 然后重启数据库, 数据库的平均负载一下子降到了 5~6。

- ☐ 发现数据库里有张 new_cheat_id 表, 读取很频繁, 而且长期处于 Sending data 状态。

怀疑是磁盘 I/O 压力过大所致, 所以操作如下:

```
explain SELECT count(new_cheat_id) FROM new_cheat WHERE account_id = '14348612'
AND offer_id = '689'\G;
```

显示结果如下所示:

```
***** 1. row *****
      id: 1
select_type: SIMPLE
      table: new_cheat
      type: ALL
possible_keys: NULL
      key: NULL
      key_len: NULL
      ref: NULL
      rows: 2529529
  Extra: Using where
1 row in set (0.00 sec)
```

上面出现的这种问题很严重, `new_cheat` 没有建好索引, 导致每次都要全表扫描 2 529 529 行记录, 严重消耗了服务器的 I/O 资源, 所以立即建好索引, 并用 `show index` 命令查看了表索引:

```
show index from new_cheat;
```

命令显示结果如下所示:

```
+-----+-----+-----+-----+-----+-----+
| Table      | Non_unique | Key_name  | Seq_in_index | Column_name | Collation |
| Cardinality | Sub_part  | Packed    | Null         | Index_type  | Comment   |
+-----+-----+-----+-----+-----+-----+
| new_cheat  | 0          | PRIMARY   | 1            | new_cheat_id | A         |
2577704 | NULL      | NULL      |              | BTREE        |           |
| new_cheat  | 1          | ip         |              |              | A         |
1288852 | NULL      | NULL      |              | BTREE        |           |
| new_cheat  | 1          | account_id |              |              | A         |
1288852 | NULL      | NULL      |              | BTREE        |           |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)
```

再来查看 `explain` 结果:

```
explain SELECT count(new_cheat_id) FROM new_cheat WHERE account_id = '14348612'
AND offer_id = '689'\G;
```

显示结果如下所示:

```
***** 1. row *****
      id: 1
select_type: SIMPLE
      table: new_cheat
      type: ref
possible_keys: account_id
      key: account_id
      key_len: 4
      ref: const
```



```
rows: 6
Extra: Using where
1 row in set (0.00 sec)
```

大家可以发现，加好了索引后，此 SQL 通过 `account_id` 索引直接读取了 6 条记录（请对比关注 `rows` 这行）就获得了查询结果，系统负载由 5~6 直接降到了 3.07~3.66 了，这个负载还是能在可接受范围内的。



注意 MySQL 的 `explain` 命令可用于 SQL 语句的查询执行计划（QEP）。这条命令的输出结果能够让我们了解 MySQL 优化器是如何执行 SQL 语句的。这条命令并没有提供任何调整建议，但它提供的重要信息能够帮助你做出调优决策。

最后要说明一点的是，对于网站来说，MySQL 单机优化对整体性能提升的作用毕竟有限，尤其是在 MySQL 单机写入方面，如果在工作中遇到了那种对 MySQL 即时写入和读取速度要求很高的场景，建议大家可以多关注分布式的 SQL 解决方案，例如 Apache 的 HBase 和 AWS 的 RedShift 等云数据仓库。

1.6 小结

本章介绍了系统架构设计的相关专业术语，以及关于 IDC 机房选择、物理服务器和 AWC EC2 类型实例的选择，最后介绍了 CentOS 6.4 x86_64 系统的优化及 MySQL 数据库的简单调优，这些都是系统架构设计的基础，希望大家能够掌握此章的内容，这对于我们以后的工作会有很大的帮助。

生产环境下的 Shell 和 Python 脚本

接触 Linux 系统十多年了，Shell 和 Python 脚本都已经完全融入笔者的生活中了。虽然 Shell 脚本只是一个简单的解释型语言，不受开发人员的重视，但对于系统运维工程师来说，它的作用举足轻重，它就像我们的瑞士军刀一样，可以帮助我们简化日常的工作并减少工作量。在系统维护工作中，Shell 脚本常常能比用 C 或 C++ 语言编写的程序更快地解决相同的问题，此外，Shell 脚本具有很好的可移植性，有时跨越 Unix 与 POSIX 兼容的系统，只须略作修改即可，甚至不必修改即可使用 Shell 脚本。不过，Shell 脚本毕竟是一门系统脚本语言，所以很多高级语言的特性它都不具备，比如面向对象和丰富的第三方类库支持，不过这一点我们可以用 Python 脚本来弥补。

2.1 Shell 和 Python 语言的简单介绍

Shell 是大家熟悉的一种脚本语言，这里简单介绍下它在日常工作中的应用。

1) 配合 Crontab 帮助我们定时执行任务，就像 MS 的计划任务一样。很多朋友向笔者反映说 Crontab 不能做秒级的计划任务，其实只要写一个 Shell 脚本，用 while..do..done 循环后放入后台执行就可以实现秒级的计划任务了。不过要记得避免出现死循环的问题。

2) 开发 Nagios 监控插件，比如我们的线上的许多业务需求都是通过 Shell 语言开发的。

3) 配合 iptables 可形成方便又安全的 iptables 脚本，还可保护主机的安全。

4) 文本过滤筛选和大数据日志分析，笔者公司的许多离线数据都是通过 Shell 配合 Python 进行分析处理的。

- 5) 可以写强大的系统性能和状态监控脚本, 并配合 Keepalived 来实现系统的高可用。
- 6) 定时备份任务和 rsync 同步重要的服务器资料, 这是 Shell 的基本功能。
- 7) 自动化安装系统环境, 规范化操作, 缩减项目实施的时间和误差。

Shell 的强大和其他未挖掘出的功能需要我们在日常工作中不断地发现和总结, 不过, 在感叹 Shell 脚本管理功能强大的同时, 也应该清楚 Shell 脚本在开发功能上的不足之处, 所以这里向大家推荐 Python, 它继承了传统编译语言的强大性和通用性, 同时也借鉴了简单脚本和解释语言的易用性, 运行速度也不慢, 适合于网站开发, 正好可以弥补 Shell 脚本的不足。

Python 是一种动态解释型的编程语言。Python 功能强大, 可支持面向对象、函数式编程, 同时还可以在 Windows、Linux 和 Unix 等多个操作系统上使用, 因此被称为“胶水语言”。Python 的简洁性、易用性使得开发过程变得简练, 特别适用于快速应用开发。笔者也发现, Python 代码在其所在公司的系统中无处不在, 在线上 Github 版本管理库中的比重也长期占据第一、第二的位置, Python 的具体应用和流行原因在后续章节里会详细说明。

2.2 Shell 编程基础

Shell 是核心程序 Kernel 之外的命令解析器, 是一个程序, 同时也是一种命令语言和程序设计语言。

作为一种命令语言 Shell 可以交互式地解析用户输入的命令。

作为一种程序设计语言 Shell 定义了各种参数, 并且提供了高级语言才有的程序控制结构, 虽然它不是 Linux 核心系统的一部分, 但是它调用了 Linux 核心的大部分功能来执行程序, 建立文件并以并行的方式来协调程序的运行。

比如, 输入命令 ls 后, Shell 就会解析 ls 这个命令并且向内核发出请求, 内核执行这个命令之后, 把结果告诉 Shell, 然后 Shell 把结果输出到屏幕。

Shell 相当于是 Windows 系统下的 command.com, 在 Windows 中这样的解析器只有一个, 但是在 Linux 中这样的解析器有很多个, 比如 Sh、Bash 和 Ksh 等。

可通过 echo \$SHELL 来查看自己运行的 Shell。在 Shell 中还可以运行子 Shell, 直接输入 csh 命令就可以进入 csh 界面了。

Linux 默认的 Shell 是 Bash, 下面的内容主要以此为主。

2.2.1 Shell 脚本的基本元素

Shell 脚本的第一行通常为如下内容:

```
#!/bin/bash //第一行
#           //表示单行注释
```

如果是多行注释呢, 应该如何操作? 多行注释如下所示:

```
:<<BLOCK
```

```
中间部分为要省略的内容
```

```
BLOCK
```

Shell 脚本的第一行均包含一个以 `#!` 为起始标志的文本行，这个特殊的起始标志表示当前文件包含一组命令，需要提交给指定的 Shell 解释执行。紧随 `#!` 标志的是一个路径名，指向执行当前 Shell 脚本文件的命令解释程序。比如：

```
#!/bin/bash
```

再比如：

```
#!/usr/bin/ruby
```

如果 Shell 脚本中包含多个特殊的标志行，那么只有一个标志行会起作用。

2.2.2 Shell 特殊字符

下面来看看 Shell 特殊字符。

- ❑ 双引号 `"`：用来使 Shell 无法认出除字符 `$`、```、`\` 之外的任何字符或字符串，也称之为弱引用。
- ❑ 单引号 `'`：用来使 Shell 无法认出所有的特殊字符，也称之为强引用。
- ❑ 反引号 ```：优先执行当前命令。
- ❑ 反斜杠 `\`：有两种作用，一种是用来使 Shell 无法认出其后的字符，使其后的字符失去特殊的含义，如有特殊含义的字符 `$`，也称为转义符。另外，如果放在指令前，有取消别名的作用，例如在 `"\rm/home/yhc/*.log"` 中，`rm` 指令前加上 `\`，作用是暂时取消别名的功能，将 `rm` 指令还原。
- ❑ 分号 `;`：允许在一行上放多个命令。
- ❑ `&`：将命令放于后台执行，建议带上 `nohup`。
- ❑ 括号 `()`：创建成组的命令。
- ❑ 大括号 `{}`：创建命令块。
- ❑ `<>&`：重定向。
- ❑ `*?[]!`：表示模式匹配。
- ❑ `$`：变量名的开头。
- ❑ `#`：表示注释（第一行除外）。
- ❑ 空格、制表符、换行符：当作空白。

2.2.3 变量和运算符

变量是放置在内存中的某个存储单元，这个存储单元里存放的是这个单元的值，这个值是可以改变的，我们称之为变量。

其中，本地变量是在用户现有的 Shell 生命周期的脚本中使用的，用户退出后变量就不

存在了, 该变量只用于该用户。

下面都是跟变量相关的命令, 这里只是大致地说明下, 后面的内容里会有详细的说明, 如下所示:

```
变量名="变量"
readonly 变量名="变量" 表示设置该变量为只读变量, 这个变量不能被改变。
echo $变量名
set 显示本地所有的变量
unset 变量名 表示清除变量
readonly 显示当前Shell下有哪些只读变量
```

环境变量用于所有用户进程 (包括子进程)。Shell 中执行的用户进程均称为子进程。不像本地变量只用于现在的 Shell。环境变量可用于所有的子进程, 它包括编辑器、脚本和应用。

环境变量主目录如下:

```
$HOME/.bash_profile (/etc/profile)
```

设置环境变量, 例句如下:

```
export test="123"
```

查看环境变量, 命令如下:

```
env
```

或者用如下命令:

```
export
```

本地变量中包含环境变量。环境变量既可以运行于父进程, 也可以运行于子进程中。本地变量则不能运行于所有的子进程中。

变量清除命令如下:

```
unset 变量名
```

再来看看位置变量, 在运行某些程序时, 程序中会带上一系列参数, 若我们要用到这些参数, 则会采用位置来表示, 对于这样的变量, 我们称之为位置变量, 目前在 Shell 中的位置变量有 10 个 (\$0~\$9), 超过 10 个则用其他方式表示, 其中, \$0 表示整个 Shell 脚本, 这点要记住。

下面举例说明位置变量的用法。比如, 有如下 test.sh 脚本内容:

```
#!/bin/bash
echo "第一个参数为": $0"
echo "第二个参数为": $1"
echo "第三个参数为": $2"
echo "第四个参数为": $3"
echo "第五个参数为": $4"
echo "第六个参数为": $5"
echo "第七个参数为": $6"
```

现在给予 test.sh 执行权限，命令如下：

```
chmod +x test.sh
./test.sh 1 2 3 4 5 6
```

命令结果显示如下：

```
第一个参数为：./test.sh
第二个参数为：1
第三个参数为：2
第四个参数为：3
第五个参数为：4
第六个参数为：5
第七个参数为：6
```

值得注意的是，从第 10 个位置参数开始，必须使用花括号括起来，如：\${10}。特殊变量 * 和 @ 表示所有的位置参数。特殊变量 # 表示位置参数的总数。

下面我们进一步详细说明下 Shell 的知识要点。

1. 运行 Shell 脚本

Shell 脚本有两种运行方式，第一种方式是利用 sh 命令，把 Shell 脚本文件名作为参数。这种执行方式要求 Shell 脚本文件具有“可读”的访问权限，然后输入 sh test.sh 即可执行。

第二种执行方式是利用 chmod 命令设置 Shell 脚本文件，使 Shell 脚本具有“可执行”的访问权限。然后直接在命令提示符下输入 Shell 脚本文件名，例如 ./test.sh。

2. 调试 Shell 脚本

使用 bash -x 可以调试 Shell 脚本，bash 会先打印出每行脚本，再打印出每行脚本的执行结果，如果只想调试其中几行脚本，可以用 set -x 和 set +x 把要调试的部分包含进来，命令如下：

```
set -x
```

脚本部分内容

```
set +x
```

这个时候可以直接运行脚本，而不需要再执行 bash -x 了。这个功能在实际工作中非常有用，可以帮助我们调试变量，找出 bug 点，总之是非常有用的功能，希望大家掌握。

3. 退出或出口状态

一个 Unix 进程或命令运行终止时，将会自动地向父进程返回一个出口状态。如果进程成功执行完毕，将会返回一个数值为 0 的出口状态。如果进程在执行过程中出现异常而未能正常结束时，将会返回一个非零值的出错代码。

在 Shell 脚本中，可以利用“exit[n]”命令在终止执行 Shell 脚本的同时，向调用脚本的父进程返回一个数值为 n 的 Shell 脚本出口状态。其中，n 必须是一个位于 0~255 范围内的整数值。如果 Shell 脚本是以不带参数的 exit 语句结束执行的，则 Shell 脚本的出口状

态就是脚本中最后执行的那条命令的出口状态。

在 Unix 系统中，为了测试一个命令或 Shell 脚本的执行结果，\$? 内部变量将返回之前执行的最后一条命令的出口状态，这些状态中，0 才是正确值，其他非零的值都表示是错误的。

4. Shell 变量

Shell 变量名可以由字母、数字和下划线等字符组成，但第一个字符必须是字母或下划线。

Shell 中的所有变量都是字符串类型的，它并不区分变量的类型，如果变量中包含下划线 (_) 的话，就要注意了，有些脚本的区别就很大，比如脚本中 \$PROJECT_svn_\$DATE.tar.gz 与 \${PROJECT}_svn_\${DATE}.tar.gz 的区别就很大，注意变量 \${PROJECT_svn}，如果不用 {} 将变量全部包括的话，Shell 则会理解成变量 \$PROJECT，后面再接着 _svn。

从用途上考虑，变量可以分为内部变量、本地变量、环境变量、参数变量和用户自定义的变量，下面分别说明它们各自的定义

- ❑ 内部变量是为了便于 Shell 编程而由 Shell 设定的变量。如错误类型的 ERRNO 变量。
- ❑ 本地变量是在代码块或函数中定义的变量，且仅在定义的范围内有效。
- ❑ 参数变量是调用 Shell 脚本或函数时传递的变量。
- ❑ 环境变量是为系统内核、系统命令和用户命令提供运行环境而设定的变量。
- ❑ 用户自定义的变量是为运行用户程序或完成某种特定的任务而设定的普通变量或临时变量。

5. 变量的赋值

变量的赋值可以采用赋值运算符 = 来实现，其语法格式如下：

```
variable=value
```

注意，赋值运算符前后不能有空格，否则会报错，写惯了 Python 后再回头写 Shell 脚本就会经常犯这种错误；未初始化的变量值为 null，使用如下变量赋值的形式，即可声明一个未初始化的变量。

如果 variable=value 赋值运算符前后有空格，则会出现如下报错信息：

```
err = 72
-bash: err: command not found
```

写惯了 Python 程序后再回头写 Shell 脚本，笔者也经常也会犯这种错误，大家不要忘了，Shell 的语法其实也是很严谨的。

6. 内部变量

Shell 提供了丰富的内部变量，为用户的 Shell 编程提供各种支持。内部变量及其意义如下所示。

- ❑ PWD：表示当前的工作目录，其变量值等同于 PWD 内部命令的输出。
- ❑ RANDOM：每次引用这个变量时，将会生成一个均匀分布的 0~32 767 范围内的随机整数。

- ❑ SCODES: 脚本已经运行的时间(秒)。
- ❑ PPID: 当前进程的父进程的进程 ID。
- ❑ \$?: 表示最近一次执行的命令或 Shell 脚本的出口状态。

7. 环境变量

主要环境变量及其意义如下所示。

- ❑ EDITOR: 用于确定命令行编辑所用的编辑程序, 通常为 vim。
- ❑ HOME: 用户主目录。
- ❑ PATH: 指定命令的检索路径。

例如, 要将 /usr/local/mysql/bin 目录添加到系统默读的 PATH 变量中, 应该如何操作呢?

```
PATH=$PATH:/usr/local/mysql/bin
export PATH
echo $PATH
```

如果想让其重启或重开一个 Shell 也生效, 又该如何操作呢?

Linux 中包含了两个重要的文件: /etc/profile 和 \$HOME/.bash_profile, 每次系统登录时都要读取这两个文件, 用来初始化系统所用到的变量, 其中 /etc/profile 是超级用户所用的, \$HOME/.bash_profile 是每个用户自己独立的, 可以通过修改该文件来设置 PATH 变量。



注意 这种方法只能使当前用户生效, 并非所有用户。

如果要让所有用户都能够用到此 PATH 变量, 可以用 vim 命令打开 /etc/profile 文件, 并在适当位置添加 PATH=\$PATH:/usr/local/mysql/bin, 然后执行 source /etc/profile 使其生效。

8. 变量的引用和替换

假定 variable 是一个变量, 在变量名字前加上 “\$” 前缀符号, 即可引用变量的值, 表示使用变量中存储的值来替换变量名字本身。

引用变量有两种形式: \$variable 与 \${variable}。



注意 位于双引号中的变量可以进行替换, 但位于单引号中的变量则不能进行替换。

9. 变量的间接引用

假定一个变量的值是另一个变量的名字, 那么根据第一个变量可以获得第三个变量的值。举例说明如下:

```
a=123
b=a
eval c=\${$b}
echo $b
echo $c
```




注意 实际工作中不推荐使用这种用法，因为写出来的脚本容易产生歧义，让人混淆，而且也不方便在团队里面交流工作。

10. 变量声明与类型定义

尽管 Shell 并未严格地区分变量的类型，但在 Bash 中，可以使用 `typeset` 或 `declare` 命令定义变量的类型，并在定义时进行初始化。

11. 部分常用命令介绍

这里将介绍工作中常用的部分 Shell 命令，如下所示。

(1) 冒号

冒号 (:) 与 `true` 语句不执行任何实际的处理动作，但可用于返回一个出口状态为 0 的测试条件。这两个语句常用于 `while` 循环结构的无限循环测试条件，在脚本中经常会见到这样的用法：

```
while :
```

这表示是一个无限循环的过程，所以使用的时候要特别注意，不要成了死循环，所以一般会定义一个 `sleep` 时间，可以实现秒级别的 `cron` 任务，其语法格式如下：

```
while :
```

```
do
```

```
    命令语句
```

```
    sleep 自己定义的秒数
```

```
done
```

(2) echo 与 print 命令

`print` 的功能与 `echo` 的功能完全一样，主要用于显示各种信息。

(3) read 命令

`read` 语句的主要功能是读取标准输入的数据，然后存储到变量参数中。如果 `read` 命令的后面有多个变量参数，则输入的数据会按空格分隔的单词顺序依次为每个变量赋值。`read` 在交互式脚本中相当有用，建议大家掌握。

`read` 命令用于接收标准输入设备（键盘）的输入，或其他文件描述符的输入（后文再详细说明）。得到输入后，`read` 命令将数据放入一个标准变量中。下面是 `read` 命令的最简单形式：

```
#!/bin/bash
```

```
echo -n "Enter your name:"
```

```
read name
```

```
echo "hello $name,welcome to my program"
```

```
exit 0
```

#参数-n的作用是不换行，echo默认是换行

#从键盘输入

#显示信息

#退出Shell程序。

由于 `read` 命令提供了 `-p` 参数，允许在 `read` 命令行中直接指定一个提示，因此上面的脚本可以简写成下面的脚本：

```
#!/bin/bash
read -p "Enter your name:" name
echo "hello $name, welcome to my program"
exit 0
```

(4) set 命令

set 命令用于修改或重新设置位置参数的值。Shell 规定，用户不能直接为位置参数赋值。使用不带参数的 set 将会输出所有的内部变量。

“set --”用于清除所有的位置参数。

(5) unset 命令

该命令用于清除 Shell 变量，把变量的值设置为 null。这个命令并不影响位置参数。

(6) expr 命令

expr 命令是一个手工命令行计数器，用于在 Linux 下求表达式变量的值，一般用于整数，也可用于字符串。其格式为：

```
expr Expression
```

expr 命令读入 Expression 参数，计算它的值，然后将结果写入到标准输出

Expression 参数应用规则如下：

- 用空格隔开每个项。
- 用 / (反斜杠) 放在 Shell 的特定字符前面。
- 对于包含空格和其他特殊字符的字符串要用引号括起来。

expr 命令支持的整数算术运算表达式如下：

- $\text{exp1}+\text{exp2}$ ，计算表达式 exp1 和 exp2 的和。
- $\text{exp1}-\text{exp2}$ ，计算表达式 exp1 和 exp2 的差。
- $\text{exp1}*\text{exp2}$ ，计算表达式 exp1 和 exp2 的乘积。
- $\text{exp1}/\text{exp2}$ ，计算表达式 exp1 和 exp2 的商。
- $\text{exp1}\%\text{exp2}$ ，计算表达式 exp1 与 exp2 的余数。

另外 expr 命令还支持字符串比较表达式，语句如下：

```
str1=str2
```

该语句是比较字符串 str1 和 str2 是否相等，如果计算结果为真，则同时输出 1，返回值为 0；反之计算结果为假，则同时输出 0，返回值为 1。

要说明的是，expr 默认是不支持浮点运算的，比如我们想在 expr 下面输出 echo "1.2*7.8" 的运算结果，那是不可能的，那么应该怎么办呢？这里可以用到 bc，举例说明如下：

```
echo "scale=2;1.2*7.8" |bc
#这里的scale用来控制小数点精度，默认为1
```

(7) let 命令

let 命令取代并扩展了 expr 命令的整数算术运算。let 命令除了支持 expr 支持的 5 种算

术运算外，还支持 +=、-=、*=、/=、%=。

12. 数值常数

Shell 脚本按十进制解释字符串中的数字字符，除非数字前有特殊的前缀或记号，若数字前有一个 0 则表示为一个八进制的数，0x 或 0X 则表示为一个十六进制的数。

13. 命令替换

命令替换的目的是获取命令的输出，且为变量赋值或对命令的输出做进一步的处理。命令替换实现的方法为采用 \$(...) 的形式引用命令或使用反向引号引用命令 'command'。如：

```
today=$(date)
echo $today
```

如果文件 filename 中包含需要删除的文件列表时，则采用如下命令：

```
rm $(cat filename)
```

14. test 语句

test 语句与 if/then 和 case 结构的语句一起，构成了 Shell 编程的控制转移结构。

test 命令的主要功能是计算紧随其后的表达式，检查文件的属性、比较字符串或比较字符串内含的整数值，然后以表达式的计算结果作为 test 命令的出口状态。如果 test 命令的出口状态为真，则返回 0；如果为假，则返回一个非 0 的数值。

test 命令的语法格式有：test expression 或 [expression]，注意方括号内侧的两边必须各有一个空格。

[[expression]] 是一种比 [expression] 更通用的测试结构，也可用于扩展 test 命令。

15. 文件测试运算符

文件测试主要指文件的状态和属性测试，其中包括文件是否存在、文件的类型、文件的访问权限及其他属性等。

下面各项为文件属性测试表达式。

- ❑ -e file，如果给定的文件存在，则条件测试的结果为真。
- ❑ -r file，如果给定的文件存在，且其访问权限是当前用户可读的，则条件测试的结果为真。
- ❑ -w file，如果给定的文件存在，且其访问权限是当前用户可写的，则条件测试的结果为真。
- ❑ -x file，如果给定的文件存在，且其访问权限是当前用户可执行的，则条件测试的结果为真。
- ❑ -s file，如果给定的文件存在，且其大于 0，则条件测试的结果为真。
- ❑ -f file，如果给定的文件存在，且是一个普通文件，则条件测试的结果为真。
- ❑ -d file，如果给定的文件存在，且是一个目录，则条件测试的结果为真。

- ❑ `-L file`, 如果给定的文件存在, 且是一个符号链接文件, 则条件测试的结果为真。
- ❑ `-c file`, 如果给定的文件存在, 且是字符特殊文件, 则条件测试的结果为真。
- ❑ `-b file`, 如果给定的文件存在, 且是块特殊文件, 则条件测试的结果为真。
- ❑ `-p file`, 如果给定的文件存在, 且是命名的管道文件, 则条件测试的结果为真。

常见代码举例如下:

```
BACKDIR=/data/backup
[ -d ${BACKDIR} ] || mkdir -p ${BACKDIR}
[ -d ${BACKDIR}/${DATE} ] || mkdir ${BACKDIR}/${DATE}
[ ! -d ${BACKDIR}/${OLDDATE} ] || rm -rf ${BACKDIR}/${OLDDATE}
```

下面是字符串测试运算符。

- ❑ `-z str`, 如果给定的字符串的长度为 0, 则条件测试的结果为真。
- ❑ `-n str`, 如果给定的字符串的长度大于 0, 则条件测试的结果为真。要求字符串必须加引号。
- ❑ `s1=s2`, 如果给定的字符串 `s1` 等同于字符串 `s2`, 则条件测试的结果为真。
- ❑ `s1!=s2`, 如果给定的字符串 `s1` 不等同于字符串 `s2`, 则条件测试的结果为真。
- ❑ `s1<s2`, 如果给定的字符串 `s1` 小于字符串 `s2`, 则条件测试的结果为真。
- ❑ `s1>s2`, 若给定的字符串 `s1` 大于字符串 `s2`, 则条件测试的结果为真。

在比较字符串的 `test` 语句中, 变量或字符串表达式的前后一定要加双引号。

再看看整数值测试运算符。`test` 语句中整数值的比较会自动采用 C 语言中的 `atoi()` 函数把字符转换成等价的 ASC 整数值。所以可以使用数字字符串和整数值进行比较。整数测试表达式为: `-eq` (等于)、`-ne` (不等于)、`-gt` (大于)、`-lt` (小于)、`-ge` (大于等于)、`-le` (小于等于)。

16. 逻辑运算符

Shell 中的逻辑运算符及其意义如下所示:

- ❑ `(expression)`: 用于计算括号中的组合表达式, 如果整个表达式的计算结果为真, 则测试结果也为真。
- ❑ `!exp`: 可对表达式进行逻辑非运算, 即对测试结果求反。例如: `test ! -f file1`。
- ❑ 符号 `-a` 或 `&&`: 表示逻辑与运算。
- ❑ 符号 `-o` 或 `||`: 表示逻辑或运算。

Shell 脚本中的用法可参考图 2-1。

指令下达情况	说明
<code>cmd1 && cmd2</code>	1. 若 <code>cmd1</code> 执行完毕且正确执行 (<code>\$?=0</code>), 则开始执行 <code>cmd2</code> 。 2. 若 <code>cmd1</code> 执行完毕且为错误 (<code>\$?≠0</code>), 则 <code>cmd2</code> 不执行。
<code>cmd1 cmd2</code>	1. 若 <code>cmd1</code> 执行完毕且正确执行 (<code>\$?=0</code>), 则 <code>cmd2</code> 不执行。 2. 若 <code>cmd1</code> 执行完毕且为错误 (<code>\$?≠0</code>), 则开始执行 <code>cmd2</code> 。

图 2-1 `&&` 与 `||` 指令说明

17. Shell 中的自定义函数

自定义函数语法比较简单，如下：

```
function 函数名()  
{  
    action;  
    [return 数值;]  
}
```

具体说明如下：

- ❑ 自定义函数既可以用带 function 参数的函数名() 定义，也可以直接用函数名() 定义，而不用带任何参数。
- ❑ 参数返回时，可以显式地加 return 返回，如果不加，则将以最后一条命令的运行结果作为返回值。return 后跟数值，取值范围 0~255。

举例说明，遍历 /usr/local/src 目录里面包含的所有文件（包括子目录），脚本内容如下：

```
#!/bin/bash  
function traverse(){  
for file in `ls $1`  
do  
    if [ -d $1/"$file" ]  
    then  
        traverse $1/"$file"  
    else  
        echo $1/"$file"  
    fi  
done  
}  
traverse "/usr/local/src"
```

18. Shell 中的数组

Shell 是支持数组的，但仅支持一维数组（不支持多维数组），并且没有限定数组的大小。类似于 C 语言，数组元素的下标由 0 开始编号。获取数组中的元素要利用下标，下标可以是整数或算术表达式，其值应大于或等于 0。

（1）定义数组

在 Shell 中，用括号来表示数组，数组元素之间用空格符号分隔开。定义数组的一般形式为：

```
array_name=(value1 ... valuen)
```

例如：

```
array_name=(value0 value1 value2 value3)
```

或者：

```
array_name=(  
value0  
value1
```

```
value2
value3
)
```

还可以单独定义数组的各个分量：

```
array_name[0]=value0
array_name[1]=value1
array_name[2]=value2
```

可以不使用连续的下标，而且下标的范围没有限制。

(2) 读取数组

读取数组元素值的一般格式为：

```
${array_name[index]}
```

例如：

```
valuen=${array_name[2]}
```

下面用一个 Shell 脚本举例说明上面的用法，脚本内容如下所示：

```
#!/bin/bash
NAME[0]="yhc"
NAME[1]="cc"
NAME[2]="gl"
NAME[3]="wendy"
echo "First Index: ${NAME[0]}"
echo "Second Index: ${NAME[1]}"
```

运行脚本，命令如下所示：

```
bash ./test.sh
```

输出结果如下所示：

```
First Index: Zara
Second Index: Qadir
```

使用 @ 或 * 可以获取数组中的所有元素，例如：

```
${array_name[*]}
${array_name[@]}
```

对上面的代码加上最后两行，如下所示：

```
echo "${NAME[*]}"
echo "${NAME[@]}"
```

运行脚本，输出：

```
First Index: yhc
Second Index: cc
yhc cc gl wendy
yhc cc gl wendy
```

(3) 获取数组的长度

获取数组长度的方法与获取字符串长度的方法相同，例如：

获取数组元素的个数，命令如下所示：

```
length=${#array_name[@]}
```

获取数组单个元素的长度，命令如下所示：

```
length=${#array_name[*]}
```

19. Shell 中的字符串截取

Shell 截取字符串的方法有很多，一般常用的有以下几种方法。

先来看第一种方法，从不同的方向截取。

从左向右截取最后一个 string 后的字符串，命令如下：

```
${variable##*string}
```

从左向右截取第一个 string 后的字符串，命令如下：

```
${variable#*string}
```

从右向左截取最后一个 string 后的字符串，命令如下：

```
${variable%%string*}
```

从右向左截取第一个 string 后的字符串，命令如下：

```
${variable%string*}
```

下面是第二种方法。

\${ 变量 :n1:n2 }：截取变量从 n1 开始的 n2 个字符，组成一个子字符串。可以根据特定字符偏移和长度，使用另一种形式的变量扩展方式来选择特定的子字符串，例如下面的命令：

```
${2:0:4}
```

这种形式的字符串截断非常简便，只须用冒号分开来指定起始字符和子字符串的长度即可，工作中用得最多的也是这种方式。

还有第三种方法。

这里利用 cut 命令来获取后缀名，命令如下：

```
ls -al | cut -d "." -f2
```

2.3 Shell 中的控制流结构

Shell 中的控制流结构也比较清晰，如下所示：

- ☐ if ...then... else...fi 语句
- ☐ case 语句
- ☐ for 循环

□ until 循环

□ while 循环

□ break 控制

□ continue 控制

工作中用得最多的就是 if 语句、for 循环、while 循环及 case 语句，大家可以以这几个为重点对象来学习。

if 语句语法如下：

```
if 条件1
then
    命令1
else
    命令2
fi
```

if 语句的进阶用法：

```
if 条件1
then
    命令1
else if 条件2
then
    命令2
else
    命令3
fi
```

举例说明下 if 语句的用法，示例如下：

```
#!/bin/bash
if [ "10" -lt "12" ]
then
    echo "10确实比12小"
else
    echo "10不小于12"
fi
```

case 语句语法如下：

```
case 值 in
模式1)
    命令1
;;
模式2)
    命令2
;;
*)
    命令3
;;
esac
```


case 取值后面必须为单词 in，每一模式必须以右括号结束。取值可以为变量或常数。匹配发现取值符合某一模式后，其间所有的命令都开始执行直至“;;”。模式匹配符“*”表示任意字符，“?”表示任意单字符，“[..]”表示类或范围中任意字符。

case 语句适合打印成绩或用于 /etc/init.d/ 服务类脚本。以下面的脚本为例来说明下。

```
#!/bin/bash
#case select
echo -n "Enter a number from 1 to 3:"
read ANS
case $ANS in
1)
    echo "you select 1"
    ;;
2)
    echo "you select 2"
    ;;
3)
    echo "you select 3"
    ;;
*)
    echo "`basename $0`: this is not between 1 and 3"
    exit;
    ;;
esac
```

下面是稍为复杂的实例说明，/etc/init.d/syslog 脚本的部分代码如下，大家注意 case 语句的用法，可以以此为参考编写自己的 case 脚本：

```
case "$1" in
start)
    start
    exit 0
    ;;
stop)
    stop
    exit 0
    ;;
reload|restart|force-reload)
    stop
    start
    exit 0
    ;;
**)
    echo "Usage: $0 {start|stop|reload|restart|force-reload}"
    exit 1
    ;;
esac
```

for 循环语句的语法如下：

```
for 变量名 in 列表
```

```
do
    命令
done
```

若变量值在列表里，则 for 循环执行一次所有命令，使用变量名访问列表并且取值。命令可为任何有效的 Shell 命令和语句，变量名为任意单词。in 列表可以包含列表、字符串和文件名，还可以是数值范围，例如 {100..200}，举例说明如下：

```
#!/bin/bash
for n in {100..200}
do
    host=192.168.1.$n
    ping -c2 $host &>/dev/null
    if [ $? = 0 ]; then
        echo "$host is UP"
    else
        echo "$host is DOWN"
    fi
done
```

while 循环语句的语法如下：

```
while 条件
do
    命令
done
```

在 Linux 中有很多逐行读取一个文件的方法，其中最常用的就是下面脚本里的方法（管道法），而且这也是效率最高、使用最多的方法，笔者最喜欢用的也是管道法。为了给大家一个直观的感受，下面将通过生成一个大文件的方式来检验各种方法的执行效率。

在脚本里，LINE 这个变量是预定义的，并不需要重新定义，\$FILENAME 后面接系统中实际存在着的文件名。

管道方法的命令语句如下：

```
cat $FILENAME | while read LINE
```

脚本举例说明如下：

```
#!/bin/bash
cat test.txt | while read LINE
do
    echo $LINE
done
}
```

2.4 sed 的基础用法及实用示例

sed 是 Linux 平台下的轻量级流编辑器，一般用于处理文本文件。sed 有许多很好的特

性。首先，它相当小巧；其次，它可以配合强大的 Shell 来完成很多复杂的功能。在笔者看来，完全可以把 sed 当作一个脚本解释器，用类似于编程的手段来完成很多事情。我们完全可以用 sed 的方式来处理日常工作中的大多数文档。它跟 vim 最大的区别在于：sed 不需要像 vim 一样打开文件，而是可以在脚本里面直接操作文档，所以大家将会发现它在 Shell 脚本里的使用频率是很高的。

2.4.1 sed 的基础语法格式

sed 的语法格式如下所示：

```
sed [-nefr] [n1,n2] 动作
```

其中：

- ❑ -n 是安静模式，只有经过 sed 处理过的行才会显示出来，其他不显示。
 - ❑ -e 表示直接在命令行模式上进行 sed 的操作。貌似是默认选项，不用写。
 - ❑ -f 将 sed 的操作写在一个文件里，用的时候 -f filename 就可以按照内容进行 sed 操作了。
 - ❑ -r 表示使 sed 支持扩展正则表达式。
 - ❑ -i 表示直接修改读取的文件内容，而不是输出到终端。
 - ❑ n1,n2 表示选择要进行处理的行，不是必需的。10,20 表示在 10~20 行之间处理。
- sed 格式中的动作支持如下参数。
- ❑ a：表示添加，后接字符串，添加到当前行的下一行。
 - ❑ c：表示替换，后接字符串，用它替换 n1 到 n2 之间的行。
 - ❑ d：表示删除符合模式的行，它的语法为 sed '/regexp/d'，// 之间是正则表达式，模式在 d 前面，d 后面一般不接任何内容。
 - ❑ i：表示插入，后接字符串，添加到当前行的上一行。
 - ❑ p：表示打印，打印选择的某个数据，通常与 -n（安静模式）一起使用。
 - ❑ s：表示搜索，还可以替换，类似于 vim 里的搜索替换功能。例如：“1,20s/old/new/g”表示将 1~20 行的 old 替换为 new，g 在这里表示处理这一行所有匹配的内容。



注意 动作最好用单引号括起来，防止因空格导致错误。

sed 的基础实例如下（下面的所有实例在 CentOS 6.4 x86_64 下已通过，这里提前将 /etc/passwd 拷贝到 /tmp 目录下）。

1) 显示 passwd 内容，将 2~5 行删除后显示，命令如下所示：

```
cat -n /tmp/passwd | sed '2,5d'
1      root:x:0:0:root:/root:/bin/bash
6      sync:x:5:0:sync:/sbin:/bin/sync
```

```

7 shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
8 halt:x:7:0:halt:/sbin:/sbin/halt
9 mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
10 uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
11 operator:x:11:0:operator:/root:/sbin/nologin
12 games:x:12:100:games:/usr/games:/sbin/nologin
13 gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
14 ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
15 nobody:x:99:99:Nobody:/:/sbin/nologin
16 vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
17 saslauth:x:499:76:"Saslauthd user":/var/empty/saslauth:/sbin/nologin
18 postfix:x:89:89:/:/var/spool/postfix:/sbin/nologin
19 sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
20 puppet:x:52:52:Puppet:/var/lib/puppet:/sbin/nologin
21 ntp:x:38:38:/:etc/ntp:/sbin/nologin
22 nagios:x:500:500:/:home/nagios:/bin/bash
23 apache:x:48:48:Apache:/var/www:/sbin/nologin
24 nginx:x:498:499:nginx user:/var/cache/nginx:/sbin/nologin

```

2) 在第2行后面的一行加上“Hello,world”字符串, 命令如下所示:

```
cat -n /tmp/passwd | sed '2a Hello,world'
```

显示结果如下所示:

```

1 root:x:0:0:root:/root:/bin/bash
2 bin:x:1:1:bin:/bin:/sbin/nologin
Hello,world
3 daemon:x:2:2:daemon:/sbin:/sbin/nologin
4 adm:x:3:4:adm:/var/adm:/sbin/nologin
5 lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
6 sync:x:5:0:sync:/sbin:/bin/sync
7 shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
8 halt:x:7:0:halt:/sbin:/sbin/halt
9 mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
10 uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
11 operator:x:11:0:operator:/root:/sbin/nologin
12 games:x:12:100:games:/usr/games:/sbin/nologin
13 gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
14 ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
15 nobody:x:99:99:Nobody:/:/sbin/nologin
16 vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
17 saslauth:x:499:76:"Saslauthd user":/var/empty/saslauth:/sbin/nologin
18 postfix:x:89:89:/:/var/spool/postfix:/sbin/nologin
19 sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
20 puppet:x:52:52:Puppet:/var/lib/puppet:/sbin/nologin
21 ntp:x:38:38:/:etc/ntp:/sbin/nologin
22 nagios:x:500:500:/:home/nagios:/bin/bash
23 apache:x:48:48:Apache:/var/www:/sbin/nologin
24 nginx:x:498:499:nginx user:/var/cache/nginx:/sbin/nologin

```

3) 在第2行后面一行加上两行字, 例如: “this is first line!” 和 “this is second line!”,

命令如下所示:

```
cat -n /tmp/passwd | sed '2a This is first line! \    //使用续航符\后按回车输入后续行
> This is second line!'    // 以' 再回车结束
```

显示结果如下所示:

```
1    root:x:0:0:root:/root:/bin/bash
2    bin:x:1:1:bin:/bin:/sbin/nologin
This is first line!
This is second line!
3    daemon:x:2:2:daemon:/sbin:/sbin/nologin
4    adm:x:3:4:adm:/var/adm:/sbin/nologin
5    lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
6    sync:x:5:0:sync:/sbin:/bin/sync
7    shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
8    halt:x:7:0:halt:/sbin:/sbin/halt
9    mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
10   uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
11   operator:x:11:0:operator:/root:/sbin/nologin
12   games:x:12:100:games:/usr/games:/sbin/nologin
13   gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
14   ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
15   nobody:x:99:99:Nobody:/:/sbin/nologin
16   vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
17   saslauth:x:499:76:"Saslauthd user":/var/empty/saslauth:/sbin/nologin
18   postfix:x:89:89:/:/var/spool/postfix:/sbin/nologin
19   sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
20   puppet:x:52:52:Puppet:/var/lib/puppet:/sbin/nologin
21   ntp:x:38:38:/:etc/ntp:/sbin/nologin
22   nagios:x:500:500:/:home/nagios:/bin/bash
23   apache:x:48:48:Apache:/var/www:/sbin/nologin
24   nginx:x:498:499:nginx user:/var/cache/nginx:/sbin/nologin
```

4) 将2~5行的内容替换成“I am a good man!”, 命令如下所示:

```
cat -n /tmp/passwd | sed '2,5c I am a good man!'
```

显示结果如下所示:

```
1    root:x:0:0:root:/root:/bin/bash
I am a good man!
6    sync:x:5:0:sync:/sbin:/bin/sync
7    shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
8    halt:x:7:0:halt:/sbin:/sbin/halt
9    mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
10   uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
11   operator:x:11:0:operator:/root:/sbin/nologin
12   games:x:12:100:games:/usr/games:/sbin/nologin
13   gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
14   ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
15   nobody:x:99:99:Nobody:/:/sbin/nologin
```

```

16  vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
17  saslauth:x:499:76:"Saslauthd user":/var/empty/saslauth:/sbin/nologin
18  postfix:x:89:89::/var/spool/postfix:/sbin/nologin
19  sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
20  puppet:x:52:52:Puppet:/var/lib/puppet:/sbin/nologin
21  ntp:x:38:38::/etc/ntp:/sbin/nologin
22  nagios:x:500:500::/home/nagios:/bin/bash
23  apache:x:48:48:Apache:/var/www:/sbin/nologin
24  nginx:x:498:499:nginx user:/var/cache/nginx:/sbin/nologin

```

5) 只显示 5~7 行, 注意 p 与 -n 的配合使用, 命令如下所示:

```
cat -n /etc/passwd | sed -n '5,7p'
```

显示结果如下所示:

```

5  lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
6  sync:x:5:0:sync:/sbin:/bin/sync
7  shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown

```

6) 使用 ifconfig 列出 IP, 我们只想要 eth0 的 IP 地址, 因此可以先用 ifconfig eth0 查看网卡 eth0 的地址, 命令如下所示:

```
ifconfig eth0
```

显示结果如下所示:

```

eth0      Link encap:Ethernet  HWaddr 00:16:3E:7F:67:C3
          inet addr:192.168.1.207  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::216:3eff:fe7f:67c3/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:11168717 errors:0 dropped:0 overruns:0 frame:0
          TX packets:83863 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:659653034 (629.0 MiB)  TX bytes:36972729 (35.2 MiB)
          Interrupt:24

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:34 errors:0 dropped:0 overruns:0 frame:0
          TX packets:34 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2716 (2.6 KiB)  TX bytes:2716 (2.6 KiB)

```

我们可以先用 grep 取出有 IP 的那一行, 然后用 sed 去掉 (替换成空) IP 前面和后面的内容, 命令如下所示:

```
ifconfig eth0 | grep "inet addr" | sed 's/^.*addr://g' | sed 's/Bcast.*$//g'
```

命令显示结果如下所示:

```
172.30.171.35
```

这里解释下这行组合命令：

grep 后面紧跟 "inet addr" 是为了单独捕获包含 IPv4 的那行内容；'^.*addr:' 表示从开头到 addr: 的字符串，将它替换为空；'Bcast.*\$' 表示从 Bcast 到结尾的串，也将它替换为空，然后就只剩下 IPv4 地址。

另外一种更简便的方法是使用 awk，命令如下所示：

```
ifconfig eth0 | grep "inet addr:" | awk -F[:] "]{print $4}"
```

命令显示结果如下所示：

```
192.168.1.207
```

awk -F[:] 的意思就是以 “:” 或空格符作为分隔符，然后打印出第 4 列，这里有些朋友可能会有疑惑：为什么不直接以如下方法来获取 IP 呢：

```
ifconfig eth0 | grep "inet addr:" | awk -F: '{print $2}'
```

大家可以看下结果，得出的结果是：

```
192.168.1.207 Bcast
```

所以还需要再进行一步操作，如下：

```
ifconfig eth0 | grep "inet addr:" | awk -F: '{print $2}' | awk '{print $1}'
```

希望大家通过这个例子好好总结一下 sed 的经典用法，第二种方法其实是 awk 的方法，awk 也是一种优秀的编辑器，现多用于截取文本字段的列。

7) 在 /etc/man.config 中，将有 man 的设置取出，但不要说明内容。命令如下所示：

```
cat /etc/man.config | grep 'MAN' | sed 's/#.*$/g' | sed '/^$/d'
```

显示结果如下所示：

```
MANPATH      /usr/man
MANPATH      /usr/share/man
MANPATH      /usr/local/man
MANPATH      /usr/local/share/man
MANPATH      /usr/X11R6/man
MANPATH_MAP  /bin          /usr/share/man
MANPATH_MAP  /sbin        /usr/share/man
MANPATH_MAP  /usr/bin      /usr/share/man
MANPATH_MAP  /usr/sbin    /usr/share/man
MANPATH_MAP  /usr/local/bin /usr/local/share/man
MANPATH_MAP  /usr/local/sbin /usr/local/share/man
MANPATH_MAP  /usr/X11R6/bin /usr/X11R6/man
MANPATH_MAP  /usr/bin/X11  /usr/X11R6/man
MANPATH_MAP  /usr/bin/mh   /usr/share/man
MANSECT      1:lp:8:2:3:3p:4:5:6:7:9:0p:n:1:p:o:1x:2x:3x:4x:5x:6x:7x:8x
```

注意, # 不一定要出现在行首。因此, /#.*\$/ 表示 # 和后面的数据 (直到行尾) 是一行注释, 将它们替换成空。/^\$/ 表示空行, 后接 d 表示删除空行。



注意 删除空行不能用替换方法, 因为空行替换成空后, 还是有换行符在那一行中。

以上就是 sed 的几种常见的语法命令, 希望大家结合下面的实例, 多在自己的机器上演练, 以尽快熟练掌握其用法。

2.4.2 sed 的用法示例

1. sed 的基础用法

1) 删除行首空格, 有下面几种方法, 代码分别如下所示:

```
sed 's/^[ ]*//g' filename
sed 's/^ *//g' filename
sed 's/^[[:space:]]*//g' filename
```

2) 在行后和行前添加新行。

行后的添加命令如下所示:

```
sed 's/pattern/&\n/g' filename
```

行前的添加命令如下所示:

```
sed 's/pattern/\n&/g' filename
```

其中, & 代表 pattern。

3) 使用变量替换 (使用双引号), 代码如下:

```
sed -e "s/$var1/$var2/g" filename
```

4) 在第一行前插入文本, 代码如下:

```
sed -i '1 i 插入字符串' filename
```

5) 在最后一行插入字符串, 代码如下:

```
sed -i '$ a 插入字符串' filename
```

6) 在匹配行前插入字符串, 代码如下:

```
sed -i '/pattern/ i "插入字符串"' filename
```

7) 在匹配行后插入字符串, 代码如下:

```
sed -i '/pattern/ a "插入字符串"' filename
```

8) 删除文本中的空行、以空格组成的行及 # 号注释的行, 代码如下:

```
grep -v ^# filename | sed /^[[[:space:]]*$/d | sed /^$/d
```


9) 将目录 /modules 下面所有文件中的 zhangsan 都修改成 list (注意备份原文件), 代码如下:

```
sed -i 's/zhangsan/list/g' `grep zhangsan -rl /modules`
```

2. 巧用 vim+sed 整理 nginxd.txt 脚本文件

笔者曾在工作中遇到过一个问题, 需要使用 Nginx 配置脚本来解决, 但在复制到服务器中并运行时发现前面的 001~100 行都有行标识符, 外带空格, 影响运行和美观。本来想逐行删除的, 后来觉得过于麻烦, 于是想到了用 sed 来解决问题, 解决方法如下。

1) 先在 vim 里删除所有行的首数字, 命令如下所示:

```
:%s/^[0-9][0-9]* //
```

2) 然后删除所有行的首空字符, 命令如下所示:

```
sed -i 's/^[[:space:]]*///' nginxd.sh
```

整个 nginxd.txt 演示脚本如下, 有兴趣的朋友也可以拿来练下手。

```
001  #!/bin/sh
002
003  # source function library
004  . /etc/rc.d/init.d/functions
005
006  # Source networking configuration.
007  . /etc/sysconfig/network
008
009  # Check that networking is up.
010  [ ${NETWORKING} = "no" ] && exit 0
011
012  RETVAL=0
013  prog="nginx"
014
015  nginxDir=/usr/local/nginx
016  nginxd=$nginxDir/sbin/nginx
017  nginxConf=$nginxDir/conf/nginx.conf
018  nginxPid=$nginxDir/nginx.pid
019
020  nginx_check()
021  {
022      if [[ -e $nginxPid ]]; then
023          ps aux |grep -v grep |grep -q nginx
024          if (( $? == 0 )); then
025              echo "$prog already running..."
026              exit 1
027          else
028              rm -rf $nginxPid &> /dev/null
029          fi
030      fi
031  }
```

```

032
033     start ()
034     {
035         nginx_check
036         if ( ( $? != 0 ) ); then
037             true
038         else
039             echo -n "$Starting $prog:"
040             daemon $nginxd -c $nginxConf
041             RETVAL=$?
042             echo
043             [ $RETVAL = 0 ] && touch /var/lock/subsys/nginx
044             return $RETVAL
045         fi
046     }
047
048     stop ()
049     {
050         echo -n "$Stopping $prog:"
051         killproc $nginxd
052         RETVAL=$?
053         echo
054         [ $RETVAL = 0 ] && rm -f /var/lock/subsys/nginx $nginxPid
055     }
056
057     reload ()
058     {
059         echo -n "$Reloading $prog:"
060         killproc $nginxd -HUP
061         RETVAL=$?
062         echo
063     }
064
065     monitor ()
066     {
067         status $prog &> /dev/null
068         if ( ( $? == 0 ) ); then
069             RETVAL=0
070         else
071             RETVAL=7
072         fi
073     }
074
075     case "$1" in
076         start)
077             start
078             ;;
079         stop)
080             stop
081             ;;
082         restart)

```

```

083         stop
084         start
085         ;;
086     reload)
087         reload
088         ;;
089     status)
090         status $prog
091         RETVAL=?
092         ;;
093     monitor)
094         monitor
095         ;;
096     *)
097         echo $"Usage: $0 {start|stop|restart|reload|status|monitor}"
098         RETVAL=1
099     esac
100     exit $RETVAL

```

脚本下载地址：<https://github.com/yuhongchun/automation>。

此文件还有很多变化，比如空格在开头，序列号在中间，也可以用 sed 来解决，不过应该写出怎样的 sed 命令来解决，就留给大家来思考吧！

3. sed 结合正则表达式批量修改文件名

笔者曾在工作中遇到了更改文件的需求，原来某文件 test.txt 中的链接地址为：

<http://www.5566.com/produce/2007080412/315613171.shtml>

<http://bz.5566.com/produce/20080808/311217.shtml>

<http://gz.5566.com/produce/20090909/311412.shtml>

现要求将 http://*.5566.com 更改为 [/home/html/www.5566.com](http://home/html/www.5566.com)，于是用 sed 结合正则表达式来解决这个问题，命令如下所示：

```
sed -i 's/http.*\.com/home/html/www.5566.com/g' test.txt
```

如果是用纯 sed 命令，方法更简单，如下所示：

```
sed -i 's@http://[^\.]*.5566.com@/home/html/www.5566.com@g' test.txt
```



注意 sed 是完全支持正则表达式的，在正则表达式里，`[^.]` 表示为非 . 的所有字符，换成 `[^/]` 也可以；另外，`@` 是 sed 的分隔符，我们也可以用其他符号，比如 `/`，但是如果要用到 `/` 的话就得表示成 `\` 了，所以笔者经常用的方法是采用 `@` 作为分隔符。

4. 在配置 .conf 文件时，为相邻的几行添加 # 号

例如，我们要将 test.txt 文件中的 31~36 行加上 # 号，使这部分内容暂时失效，该如何实现呢？

在 vim 中, 可以执行如下命令:

```
:31,36 s/^/#/
```

而使用 sed 执行更加方便, 命令如下所示:

```
sed -i '31,36s/^/#/' test.txt
```

反之, 如果要将 31~36 行带 # 号的全部删除, 用 sed 又该如何实现呢? 方法如下:

```
sed -i '31,36s/^#//' test.txt
```

许多人习惯在这个方法后面带个 g, 事实上, 如果没有 g, 则表示从行的左端开始匹配, 每一行第一个与之匹配的都会被换掉; 如果有 g, 则表示每一行所有与之匹配的都会被换掉。

5. 利用 sed 分析日志

利用 sed 还可以很方便地分析日志。例如, 在以下的 secure 日志文件中, 想用 sed 抓取 12:48:48 至 12:48:55 的日志:

```
Apr 17 05:01:20 localhost sshd[16375]: pam_unix(sshd:auth): authentication
failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=222.186.37.226 user=root
Apr 17 05:01:22 localhost sshd[16375]: Failed password for root from
222.186.37.226 port 60700 ssh2
Apr 17 05:01:22 localhost sshd[16376]: Received disconnect from 222.186.37.226:
11: Bye Bye
Apr 17 05:01:22 localhost sshd[16377]: pam_unix(sshd:auth): authentication
failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=222.186.37.226 user=root
Apr 17 05:01:24 localhost sshd[16377]: Failed password for root from
222.186.37.226 port 60933 ssh2
Apr 17 05:01:24 localhost sshd[16378]: Received disconnect from 222.186.37.226:
11: Bye Bye
Apr 17 05:01:24 localhost sshd[16379]: pam_unix(sshd:auth): authentication
failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=222.186.37.226 user=root
Apr 17 05:01:26 localhost sshd[16379]: Failed password for root from
222.186.37.226 port 32944 ssh2
Apr 17 05:01:26 localhost sshd[16380]: Received disconnect from 222.186.37.226:
11: Bye Bye
Apr 17 05:01:27 localhost sshd[16381]: pam_unix(sshd:auth): authentication
failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=222.186.37.226 user=root
Apr 17 05:01:29 localhost sshd[16381]: Failed password for root from
222.186.37.226 port 33174 ssh2
Apr 17 05:01:29 localhost sshd[16382]: Received disconnect from 222.186.37.226:
11: Bye Bye
Apr 17 05:01:29 localhost sshd[16383]: pam_unix(sshd:auth): authentication
failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=222.186.37.226 user=root
Apr 17 05:01:31 localhost sshd[16383]: Failed password for root from
222.186.37.226 port 33474 ssh2
Apr 17 05:01:31 localhost sshd[16384]: Received disconnect from 222.186.37.226:
11: Bye Bye
```



```
Apr 17 05:01:32 localhost sshd[16385]: pam_unix(sshd:auth): authentication
failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=222.186.37.226 user=root
```

可以利用 `sed` 截取日志命令，如下所示：

```
cat /var/log/secure | sed -n '/12:48:48/,/12:48:55/p'
```

脚本显示结果如下所示：

```
Apr 23 12:48:48 localhost sshd[20570]: Accepted password for root from 220.249.72.138
port 27177 ssh2
Apr 23 12:48:48 localhost sshd[20570]: pam_unix(sshd:session): session opened
for user root by (uid=0)
Apr 23 12:48:55 localhost sshd[20601]: Accepted password for root from 220.249.72.138
port 59754 ssh2
```

`sed` 的用法还有许多，这就要靠大家在日常工作中归纳总结了。有兴趣的朋友还可以多了解下 `awk` 的用法，我们在工作中要频繁地分析日志文件，`awk+sed` 是比较好的选择，下面介绍下 `awk` 的基本使用方法。

2.5 awk 的基础用法及实用示例

1. awk 工具简介

`awk` 是一个强大的文本分析工具，相对于 `grep` 的查找、`sed` 的编辑，`awk` 在对数据进行分析并生成报告时，显得尤为强大。简单来说，`awk` 就是把文件逐行地读入，然后以空格为默认分隔符将每行进行切片，切开的部分再进行各种分析处理。`awk` 的名称得自于它的创始人 Alfred Aho、Peter Weinberger 和 Brian Kernighan 姓氏的首个字母。实际上 `awk` 的确拥有自己的语言：`awk` 程序设计语言，三位创建者已将它正式定义为“样式扫描和处理语言”。

`awk` 允许我们创建简短的程序，这些程序可读取输入文件、为数据排序、处理数据、对输入执行计算及生成报表，还有无数其他的功能。

2. 使用方法

`awk` 的命令格式如下：

```
awk 'pattern {action}' filename
```

其中，`pattern` 就是要表示的正则表达式，它表示 `awk` 在数据中查找的内容，而 `action` 是在找到匹配内容时所执行的一系列命令。

`awk` 语言的最基本功能是在文件或字符串中基于指定的规则浏览和抽取信息，在抽取信息后，才能进行其他文本操作。完整的 `awk` 脚本通常用来格式化文本文件中的信息。

通常，`awk` 是以文件的一行为处理单位的。`awk` 每接收文件的一行后，就会执行相应的命令来处理文本。

下面介绍一下 `awk` 程序设计模型。

`awk` 程序由 3 部分组成，分别为：

初始化 (处理输入前做的准备，放在 `BEGIN` 块中)

数据处理 (处理输入数据)

收尾处理 (处理输入完成后要进行的处理，放到 `END` 块中)

其中，在“数据处理”过程中，指令被写成一系列模式 / 动作过程，模式用于测试输入行的规则，以确定是否将规则应用于这些输入行。

3. `awk` 调用方式

`awk` 主要有三种调用方式，下面分别来看看。

(1) 命令行方式

```
awk [-F field-separator] 'commands' filename
```

其中，`commands` 是真正的 `awk` 命令，`[-F 域分隔符]` 是可选的，`filename` 是待处理的文件。

在 `awk` 文件的各行中，由域分隔符分开的每一项称为一个域。通常，在不指名 `-F` 域分隔符的情况下，默认的域分隔符是空格。

(2) 使用 `-f` 选项调用 `awk` 程序

`awk` 允许将一段 `awk` 程序写入一个文本文件中，然后在 `awk` 命令行中用 `-f` 选项调用并执行这段程序，命令如下：

```
awk -f awk-script-file filename
```

其中，`-f` 选项加载 `awk-script-file` 中的 `awk` 脚本，`filename` 表示文件名。

(3) 利用命令解释器调用 `awk` 程序

利用 Linux 系统支持的命令解释器功能可以将一段 `awk` 程序写入文本文件中，然后在它的第一行加上 `#!/bin/awk -f`。

4. `awk` 详细语法

与其他 Linux 命令一样，`awk` 拥有自己的语法：

```
awk [-F re] [parameter...] ['prog'] [-f progfile][in_file...]
```

其中，

❑ `-F re`：允许 `awk` 更改其字段分隔符。

❑ `parameter`：该参数帮助为不同的变量赋值。

❑ `prog`：`awk` 的程序语句段。这个语句段必须用单引号 ' 和 ' 括起，以防被 Shell 解释。前面已经提到过这个程序语句段的标准形式，如下所示：

```
awk 'pattern {action}' filename
```

其中 `pattern` 参数可以是 `egrep` 正则表达式中的任何一个，它可以使用语法 `/re/` 再加上

一些样式匹配技巧构成。与 sed 类似，也可以使用“,”分开两种样式以选择某个范围。

action 参数总是被大括号包围，它由一系列 awk 语句组成，各语句之间用“;”分隔。awk 会解释它们，并在 pattern 给定的样式匹配记录上执行相关操作。

事实上，在使用该命令时可以省略 pattern 和 action 两者中的某一个，但不能两者同时省略。省略 pattern 表示没有样式匹配，对所有行（记录）均执行操作；省略 action 表示执行默认的操作——在标准输出上显示。

❑ -f progfile：允许 awk 调用并执行 progfile 指定的程序文件。progfile 是一个文本文件，它必须符合 awk 的语法。

❑ in_file：awk 的输入文件，awk 允许对多个输入文件进行处理。值得注意的是 awk 不修改输入文件。

如果未指定输入文件，awk 将接受标准输入，并将结果显示在标准输出上。

5. awk 脚本编写

(1) awk 的内置变量

awk 的内置变量主要有如下几种。

❑ FS：输入数据的字段分隔符。

❑ RS：输入数据的记录分隔符。

❑ OFS：输出数据的字段分隔符。

❑ ORS：输出数据的记录分隔符。另一类是系统自动改变的，比如：NF 表示当前记录的字段个数，NR 表示当前记录的编号等。

举个例子，可用如下命令打印 passwd 中的第 1 个和第 3 个字段：

```
awk -F ":" '{ print $1 " " $3 }' /tmp/passwd
```

(2) pattern/action 模式

awk 程序部分采用了 pattern/action 模式，即针对匹配 pattern 的数据，使用 action 逻辑进行处理。来看下面这两个例子。

判断当前是不是空格，命令如下：

```
/^$/ {print "This is a blank line!"}
```

判断第 5 个字段是否含有“MA”，命令如下：

```
$5 ~ /MA/ {print $1 " " $3}
```

(3) awk 与 Shell 混用

因为 awk 可以作为一个 Shell 命令使用，因此 awk 能与 Shell 脚本程序很好地融合在一起，这点为实现 awk 与 Shell 程序的混合编程提供了可能。实现混合编程的关键是 awk 与 Shell 脚本之间的对话，换言之，就是 awk 与 Shell 脚本之间的信息交流：awk 从 Shell 脚本中获取所需的信息（通常是变量的值）、在 awk 中执行 Shell 命令行、Shell 脚本将命令执行

的结果送给 awk 处理, 以及 Shell 脚本读取 awk 的执行结果等, 另外需要注意的是在 Shell 脚本中读取 awk 变量的方式, 一般是通过 "\$ 变量名" 的方式来读取 Shell 程序中的变量。

6. awk 内置变量

awk 有许多内置变量用于设置环境信息, 这些变量可以被改变, 下面列举了工作中最常用的一些 awk 变量, 变量及其意义如下所示:

ARGC	命令行参数个数
ARGV	命令行参数排列
ENVIRON	支持队列中系统环境变量的使用
FILENAME	awk 浏览的文件名
FNR	浏览文件的记录数
FS	设置输入域分隔符, 等价于命令行 -F 选项
NF	浏览记录的域的个数
NR	已读的记录数
OFS	输出域分隔符
ORS	输出记录分隔符
RS	控制记录分隔符

此外, \$0 变量是指整条记录。\$1 表示当前行的第一个域, \$2 表示当前行的第二个域……依次类推。

7. awk 中的 print 和 printf

awk 中同时提供了 print 和 printf 两种用于打印输出的函数。

其中 print 函数的参数可以是变量、数值或字符串。字符串必须用双引号引用, 参数用逗号分隔。如果没有逗号, 参数就会串联在一起而无法区分。这里, 逗号的作用与输出文件的分隔符的作用是一样的, 只是后者是空格而已。

printf 函数, 其用法和 C 语言中 printf 函数基本相似, 可以格式化字符串, 输出复杂结果时, printf 的显示结果更加人性化。

使用示例如下所示:

```
awk -F ':' '{printf("filename:%10s,linenumber:%s,columns:%s,linecontent:%s\n",FILENAME,NR,NF,$0)}' /tmp/passwd
```

命令显示结果如下所示:

```
filename:/tmp/passwd,linenumber:1,columns:7,linecontent:root:x:0:0:root:/root:/bin/bash
filename:/tmp/passwd,linenumber:2,columns:7,linecontent:bin:x:1:1:bin:/bin:/sbin/nologin
filename:/tmp/passwd,linenumber:3,columns:7,linecontent:daemon:x:2:2:daemon:/sbin:/sbin/
nologin
filename:/tmp/passwd,linenumber:4,columns:7,linecontent:adm:x:3:4:adm:/var/adm:/sbin/
nologin
filename:/tmp/passwd,linenumber:5,columns:7,linecontent:lp:x:4:7:lp:/var/spool/lpd:/
sbin/nologin
filename:/tmp/passwd,linenumber:6,columns:7,linecontent:sync:x:5:0:sync:/sbin:/bin/sync
filename:/tmp/passwd,linenumber:7,columns:7,linecontent:shutdown:x:6:0:shutdown:/sbin:/
sbin/shutdown
```



```

filename:/tmp/passwd,linenumber:8,columns:7,linecontent:halt:x:7:0:halt:/sbin:/sbin/halt
filename:/tmp/passwd,linenumber:9,columns:7,linecontent:mail:x:8:12:mail:/var/
    spool/mail:/sbin/nologin
filename:/tmp/passwd,linenumber:10,columns:7,linecontent:uucp:x:10:14:uucp:/var/
    spool/uucp:/sbin/nologin
filename:/tmp/passwd,linenumber:11,columns:7,linecontent:operator:x:11:0:operat
    or:/root:/sbin/nologin
filename:/tmp/passwd,linenumber:12,columns:7,linecontent:games:x:12:100:games:/
    usr/games:/sbin/nologin
filename:/tmp/passwd,linenumber:13,columns:7,linecontent:gopher:x:13:30:gopher:/
    var/gopher:/sbin/nologin
filename:/tmp/passwd,linenumber:14,columns:7,linecontent:ftp:x:14:50:FTP User:/
    var/ftp:/sbin/nologin
filename:/tmp/passwd,linenumber:15,columns:7,linecontent:nobody:x:99:99:Nobody:/:/
    sbin/nologin
filename:/tmp/passwd,linenumber:16,columns:7,linecontent:vcsa:x:69:69:virtual
    console memory owner:/dev:/sbin/nologin
filename:/tmp/passwd,linenumber:17,columns:7,linecontent:saslauth:x:499:76:"Saslauthd
    user":/var/empty/saslauth:/sbin/nologin
filename:/tmp/passwd,linenumber:18,columns:7,linecontent:postfix:x:89:89::/var/
    spool/postfix:/sbin/nologin
filename:/tmp/passwd,linenumber:19,columns:7,linecontent:sshd:x:74:74:Privilege-
    separated SSH:/var/empty/sshd:/sbin/nologin
filename:/tmp/passwd,linenumber:20,columns:7,linecontent:puppet:x:52:52:Puppet:/
    var/lib/puppet:/sbin/nologin
filename:/tmp/passwd,linenumber:21,columns:7,linecontent:ntp:x:38:38::/etc/ntp:/
    sbin/nologin
filename:/tmp/passwd,linenumber:22,columns:7,linecontent:nagios:x:500:500::/
    home/nagios:/bin/bash
filename:/tmp/passwd,linenumber:23,columns:7,linecontent:apache:x:48:48:Apache:/
    var/www:/sbin/nologin
filename:/tmp/passwd,linenumber:24,columns:7,linecontent:nginx:x:498:499:nginx
    user:/var/cache/nginx:/sbin/nologin

```

参考文档 <http://blog.pengduncun.com/?p=876>。

8. 工作示例

截取出 init 中 PID 的示例命令如下：

```
ps -aux | grep init | grep -v grep | awk '{print $2}'
```

截取网卡 eth0 的 IPv4 地址，示例命令如下：

```
ifconfig eth0 | grep "inet addr:" | awk -F: '{print $2}' | awk '{print $1}'
```

找出当前系统的自启动服务，示例命令如下：

```
chkconfig --list | grep 3:on | awk '{print $1}'
```

取出 vmstat 第 4 项的平均值，示例命令如下：

```
vmstat 1 4 | egrep -v "io|free" | awk '{sum+=$4} END{print sum/4}'
```

以 | 为分隔符, 汇总 /yundisk/log/hadoop/ 下的 hadoop 第 9 项日志并打印, 示例命令如下:

```
cat /yundisk/log/hadoop/hadoop_clk_*.log | awk -F '|' 'BEGIN{count=0} $2>0
{count=count+$9} END {print count}'
```

2.6 生产环境下的 Shell 和 Python 脚本分类

生产环境下的 Shell 和 Python 脚本的作用还是挺多的, 这里根据 2.1 节所介绍的日常工作中 Shell 脚本的作用, 将生产环境下的 Shell 脚本分为备份类、监控类、统计类、运维开发类和自动化运维类。前面 3 类从字面意义上看比较容易理解, 后面的两类需要稍微解释一下, 运维开发类脚本是利用 Shell 或 Python 实现一些非系统类的管理工作, 比如 SVN 的发布程序等; 而自动化运维类脚本则是利用 Shell 或 Python 来自动替我们做一些烦琐的工作, 比如自动生成并分配密码给开发组的用户, 或者自动安装 LNMP 环境等。下面会就这些分类列举一些具体的实例, 以便于大家理解。另外值得说明的一点是, 这些实例都源自于笔者的线上环境, 大家拿过来稍微改动一下 IP 或备份一下目录基本上就可以直接使用了。

另外, 因为现在的线上业务大多采用的是 AWS EC2 机器, 基本上采用的都是 Amazon Linux 系统, 所以这里先跟大家简单介绍下 Amazon Linux 系统。

Amazon Linux 系统由 Amazon Web Services (AWS) 提供, 旨在为 Amazon EC2 上运行的应用程序提供稳定、安全、高性能的执行环境。此外, 它还包括能够与 AWS 轻松集成的软件包, 比如启动配置工具和许多常见的 AWS 库及工具等。AWS 为运行 Amazon Linux 系统的所有实例提供持续的安全性和维护更新。

(1) 启动并连接到 Amazon Linux 实例

要启动 Amazon Linux 实例, 请使用 Amazon Linux AMI (映像)。AWS 向 Amazon EC2 用户提供 Amazon Linux AMI, 无需额外费用。找到需要的 AMI 后, 记下 AMI ID, 然后就可以使用 AMI ID 来启动并连接到相应的实例了。

默认情况下, Amazon Linux 不支持远程 root SSH。此外, 密码验证已禁用, 以防止强力 (brute-force) 密码攻击。要在 Amazon Linux 实例上启用 SSH 登录, 必须在实例启动时为其提供密钥对, 还必须设置用于启动实例的安全组以允许 SSH 访问。默认情况下, 唯一可以使用 SSH 进行远程登录的账户是 ec2-user; 此账户还拥有 sudo 特权。如果希望启动远程根登录, 请注意, 其安全性不及依赖密钥对和二级用户。

有关启动和使用 Amazon Linux 实例的信息, 请参阅启动实例。有关连接到 Amazon Linux 实例的更多信息, 请参阅连接到 Linux 实例。

(2) 识别 Amazon Linux AMI 映像

每个映像都包含唯一的 /etc/image-id, 用于识别 AMI。此文件包含了有关映像的信息。

下面是 /etc/image-id 文件示例，命令如下：

```
cat /etc/image-id
```

命令显示结果如下所示：

```
image_name="amzn-ami-hvm"  
image_version="2015.03"  
image_arch="x86_64"  
image_file="amzn-ami-hvm-2015.03.0.x86_64.ext4.gpt"  
image_stamp="366c-fff6"  
image_date="20150318153038"  
recipe_name="amzn ami"  
recipe_id="1c207c1f-6186-b5c9-4e1b-9400-c2d8-a3b2-3d11fdf8"
```

其中，image_name、image_version 和 image_arch 项目来自 Amazon 用于构建映像的配方。image_stamp 只是映像创建期间随机生成的唯一十六进制值。image_date 项目的格式为 YYYYMMDDhhmmss，是映像创建时的 UTC 时间。recipe_name 和 recipe_id 是 Amazon 用于构建映像的构建配方的名称和 ID，用于识别当前运行的 Amazon Linux 的版本。从 yum 存储库安装更新时，此文件不会更改。

Amazon Linux 包含 /etc/system-release 文件，用于指定当前安装的版本。此文件通过 yum 进行更新，是 system-release RPM 的一部分。

下面是 /etc/system-release 文件示例，命令如下：

```
cat /etc/system-release
```

命令显示结果如下所示：

```
Amazon Linux AMI release 2015.03
```



说明 Amazon Linux 系统这部分内容摘录自 http://docs.aws.amazon.com/zh_cn/AWSEC2/latest/UserGuide/AmazonLinuxAMIBasics.html#IdentifyingLinuxAMI_Images

2.6.1 备份类脚本

俗话说得好，备份是救命的稻草。特别是重要的数据和代码，这些都是公司的重要资产，所以备份是必须的。备份能在我们不慎执行了一些毁灭性的工作之后（比如不小心删除了数据），进行恢复工作。许多有实力的公司在国内好几个地方都设有灾备机房，而且用的都是价格不菲的 EMC 高端存储。可能会有朋友要问：如果我们没有存储怎么办？这点可以参考一下笔者公司的备份策略，即：在执行本地备份的同时，让 Shell 脚本自动上传数据到另一台 FTP 备份服务器中，这种异地备份策略成本比较小，无须存储，而且安全系统高，相当于双备份，本地和异地同时出现数据损坏的概率几乎是不可能的。

此双备份策略的具体步骤如下。

首先，做好准备工作。先安装一台 CentOS 6.4 x86_64 的备份服务器，并安装 vsftpd

服务,稍微改动一下配置后启动。另外,关于 vsftpd 的备份目录,可以选择做 RAID1 或 RAID5 的分区或存储。

vsftpd 服务的安装如下, CentOS 6.4 x86_64 下自带的 yum 极为方便。

```
yum -y install vsftpd
service vsftpd start
chkconfig vsftpd on
```

vsftpd 的配置比较简单,详细语法略过,这里只给出配置文件,可以通过组合使用如下命令直接得出 vsftpd.conf 中有效的文件内容:

```
grep -v "^#" /etc/vsftpd/vsftpd.conf | grep -v '^\$'
local_enable=YES
write_enable=YES
local_umask=022
dirmessage_enable=YES
xferlog_enable=YES
connect_from_port_20=YES
xferlog_std_format=YES
listen=YES
chroot_local_user=YES
pam_service_name=vsftpd
userlist_enable=YES
tcp_wrappers=YES
```

chroot_local_user=YES 这条语句需要重点强调一下。它的作用是对用户登录权限进行限制,即所有本地用户登录 vsftpd 服务器时只能在自己的家目录下,这是基于安全的考虑,笔者在编写脚本的过程中也考虑到了这点,如果大家要移植此脚本到自己的工作环境中,不要忘了这条语句,不然的话异地备份极有可能会失效。

另外,我们在备份服务器上应该建立备份用户,例如 svn,并为其分配密码,还应该将其家目录更改为备份目录,即 /data/backup/svn-bakcup,这样的话更方便备份工作,以下备份脚本依此类推。

1. 版本控制软件 SVN 代码库的备份脚本

版本控制软件 SVN 的重要性在这里就不再多说了,现在很多公司基本还是利用 SVN 作为提交代码集中管理的工具,所以做好其备份工作的重要性就不言而喻了。这里的轮询周期为 30 天一次,Shell 会自动删除 30 天以前的文件。在 vsftpd 服务器上建立相应的备份用户 svn 的脚本内容如下(此脚本在 CentOS 5.8 x86_64 下已测试通过):

```
#!/bin/sh
SVNDIR=/data/svn
SVNADMIN=/usr/bin/svnadmin
DATE=`date +%Y-%m-%d`
OLDDATE=`date +%Y-%m-%d -d '30 days'`
BACKDIR=/data/backup/svn-backup
```



```

[ -d ${BACKDIR} ] || mkdir -p ${BACKDIR}
LogFile=${BACKDIR}/svnbak.log
[ -f ${LogFile} ] || touch ${LogFile}
mkdir ${BACKDIR}/${DATE}

for PROJECT in myproject official analysis mypharma
do
    cd $SVNDIR
    $SVNADMIN hotcopy $PROJECT $BACKDIR/$DATE/$PROJECT --clean-logs
    cd $BACKDIR/$DATE
    tar zcvf ${PROJECT}_svn_${DATE}.tar.gz $PROJECT > /dev/null
    rm -rf $PROJECT
    sleep 2
done

HOST=192.168.2.112
FTP_USERNAME=svn
FTP_PASSWORD=svn101

cd ${BACKDIR}/${DATE}

ftp -i -n -v << !
open ${HOST}
user ${FTP_USERNAME} ${FTP_PASSWORD}
bin
cd ${OLDDATE}
mdelete *
cd ..
rmdir ${OLDDATE}
mkdir ${DATE}
cd ${DATE}
mput *
bye
!

```

2. MySQL 从数据库备份脚本

主 MySQL 脚本跟其比较类似，主要是用 MySQL 自带的命令 `mysqldump` 进行备份的。这里要说明的是，本地的轮询周期为 20 天，`vsftpd` 的轮询周期为 60 天，备份后就算是用 `gzip` 压缩，MySQL 的数据库还是很大，大家可以根据实际需求来更改这个时间，注意不要让数据库撑爆你的备份服务器。由于是内部开发环境，密码设置得比较简单，如果要将其移植到自己的公网服务器上，就要在安全方面多注意一下了。另外附带说明的是，`--opt` 只是一个快捷选项，等同于同时添加 `--add-drop-tables` `--add-locking` `--create-option` `--disable-keys` `--extended-insert` `--lock-tables` `--quick` `--set-charset` 选项。本选项能让 `mysqldump` 很快地导出数据，并且导出的数据可以很快被导回。该选项默认是开启的，可以用 `--skip-opt` 将其禁用。注意，如果运行 `mysqldump` 没有指定 `--quick` 或 `--opt` 选项，则会将整个结果集放

在内存中。如果导出的是大数据库则可能会出现问题，脚本内容如下（此脚本在 CentOS 5.8 x86_64 下已测试通过）：

```
#!/bin/bash
USERNAME=mysqlbackup
PASSWORD=mysqlbackup
DATE=`date +%Y-%m-%d`
OLDDATE=`date +%Y-%m-%d -d '-20 days'`
FTPOLDDATE=`date +%Y-%m-%d -d '-60 days'`
MYSQL=/usr/local/mysql/bin/mysql
MYSQLDUMP=/usr/local/mysql/bin/mysqldump
MYSQLADMIN=/usr/local/mysql/bin/mysqladmin
SOCKET=/tmp/mysql.sock
BACKDIR=/data/backup/db

[ -d ${BACKDIR} ] || mkdir -p ${BACKDIR}
[ -d ${BACKDIR}/${DATE} ] || mkdir ${BACKDIR}/${DATE}
[ ! -d ${BACKDIR}/${OLDDATE} ] || rm -rf ${BACKDIR}/${OLDDATE}

for DBNAME in mysql test report
do
    ${MYSQLDUMP} --opt -u${USERNAME} -p${PASSWORD} -S${SOCKET} ${DBNAME} | gzip >
        ${BACKDIR}/${DATE}/${DBNAME}-backup-${DATE}.sql.gz
    echo "${DBNAME} has been backup successful"
    /bin/sleep 5
done

HOST=192.168.4.45
FTP_USERNAME=dbmysql
FTP_PASSWORD=dbmysql
cd ${BACKDIR}/${DATE}
ftp -i -n -v << !
open ${HOST}
user ${FTP_USERNAME} ${FTP_PASSWORD}
bin
cd ${FTPOLDDATE}
mdelete *
cd ..
rmdir ${FTPOLDDATE}
mkdir ${DATE}
cd ${DATE}
mput *
bye
!
```

3. MySQL 数据备份至 S3 文件系统

这里先来介绍下亚马逊的分布式文件系统 S3，S3 为开发人员提供了一个高可扩展（Scalability）、高持久性（Durability）和高可用（Availability）的分布式数据存储服务。它是一个完全针对互联网的数据存储服务，借助一个简单的 Web 服务接口就可以通过互联网

在任何时候访问 S3 上的数据。当然存放在 S3 上的数据要可以进行访问控制以保障数据的安全性。这里所说的访问 S3 包括读、写、删除等多种操作。在脚本的最后，采用 AWS S3 命令中的 cp 将 MySQL 上传至 s3://example-shar 这个 bucket 上面（S3 详细资料介绍 <http://aws.amazon.com/cn/s3/>），脚本内容如下所示（此脚本在 Amazon Linux AMI x86_64 下已测试通过）：

```
#!/bin/bash
#
# Filename:
# backupdatabase.sh
# Description:
# backup cms database and remove backup data before 7 days
# crontab
# 55 23 * * * /bin/sh /yundisk/cms/crontab/backupdatabase.sh >> /yundisk/cms/
# crontab/backupdatabase.log 2>&1

DATE=`date +%Y-%m-%d`
OLDDATE=`date +%Y-%m-%d -d '-7 days'`

#MYSQL=/usr/local/mysql/bin/mysql
#MYSQLDUMP=/usr/local/mysql/bin/mysqldump
#MYSQLADMIN=/usr/local/mysql/bin/mysqladmin

BACKDIR=/yundisk/cms/database
[ -d ${BACKDIR} ] || mkdir -p ${BACKDIR}
[ -d ${BACKDIR}/${DATE} ] || mkdir ${BACKDIR}/${DATE}
[ ! -d ${BACKDIR}/${OLDDATE} ] || rm -rf ${BACKDIR}/${OLDDATE}

mysqldump --default-character-set=utf8 --no-autocommit --quick --hex-lob --single-
transaction -uroot cms_production | gzip > ${BACKDIR}/${DATE}/cms-backup-
${DATE}.sql.gz
echo "Database cms_production and bbs has been backup successful"
/bin/sleep 5

aws s3 cp ${BACKDIR}/${DATE}/* s3://example-share/cms/databackup/
```

2.6.2 统计类脚本

统计工作一直是 Shell 和 Python 脚本的强项，我们完全可以利用 sed、awk 再加上正则表达式，写出强大的统计脚本来分析我们的系统日志、安全日志及服务器应用日志等。

1. Nginx 负载均衡器日志汇总脚本

以下脚本是用来分析 Nginx 负载均衡器的日志的，作为 Awstats 的补充，它可以快速得出排名最前的网站和 IP 等，脚本内容如下（此脚本在 CentOS 5.8/6.4 x86_64 下均已测试通过）：

```
#!/bin/bash
```

```

if [ $# -eq 0 ]; then
    echo "Error: please specify logfile."
    exit 0
else
    LOG=$1
fi

if [ ! -f $1 ]; then
    echo "Sorry, sir, I can't find this apache log file, pls try again!"
    exit 0
fi

#####
echo "Most of the ip:"
echo "-----"
awk '{ print $1 }' $LOG | sort | uniq -c | sort -nr | head -10
echo
echo
#####
echo "Most of the time:"
echo "-----"
awk '{ print $4 }' $LOG | cut -c 14-18 | sort | uniq -c | sort -nr | head -10
echo
echo
#####
echo "Most of the page:"
echo "-----"
awk '{print $11}' $LOG | sed 's/^.*\(.cn*\)\"/\1/g' | sort | uniq -c | sort -rn | head -10
echo
echo
#####
echo "Most of the time / Most of the ip:"
echo "-----"
awk '{ print $4 }' $LOG | cut -c 14-18 | sort -n | uniq -c | sort -nr | head -10 > timelog

for i in `awk '{ print $2 }' timelog`
do
    num=`grep $i timelog | awk '{ print $1 }'`
    echo " $i $num"
    ip=`grep $i $LOG | awk '{ print $1 }' | sort -n | uniq -c | sort -nr | head -10`
    echo "$ip"
    echo
done
rm -f timelog

```

2. 探测多节点 Web 服务质量

工作中有时会出现网络延迟导致程序返回数据不及时的问题,这时就需要精准定位机器是在哪个时间段出现了网络延迟的情况。对此,可以通过 Python 下的 pycurl 模块来实现定位,它可以通过调用 pycurl 提供的方法,来探测 Web 服务质量,比如了解相应的 HTTP

状态码、请求延时、HTTP 头信息、下载速度等，脚本内容如下所示（此脚本在 Amazon Linux AMI x86_64 下已测试通过）：

```
#!/usr/bin/python
#encoding:utf-8
#*/30 * * * * /usr/bin/python /root/dnstime.py >> /root/myreport.txt 2>&1

import os
import time
import sys
import pycurl
#import commands
import time

URL="http://imp-east.example.net"
ISOTIMEFORMAT="%Y-%m-%d %X"
c = pycurl.Curl()
c.setopt(pycurl.URL, URL)
c.setopt(pycurl.CONNECTTIMEOUT, 5)
c.setopt(pycurl.TIMEOUT, 5)
c.setopt(pycurl.FORBID_REUSE, 1)
c.setopt(pycurl.MAXREDIRS, 1)
c.setopt(pycurl.NOPROGRESS, 1)
c.setopt(pycurl.DNS_CACHE_TIMEOUT, 30)
indexfile = open(os.path.dirname(os.path.realpath(__file__))+"/content.txt", "wb")
c.setopt(pycurl.WRITEHEADER, indexfile)
c.setopt(pycurl.WRITEDATA, indexfile)
try:
    c.perform()
except Exception,e:
    print "conneccion error:"+str(e)
    indexfile.close()
    c.close()
    sys.exit()

NAMELOOKUP_TIME = c.getinfo(c.NAMELOOKUP_TIME)
CONNECT_TIME = c.getinfo(c.CONNECT_TIME)
PRETRANSFER_TIME = c.getinfo(c.PRETRANSFER_TIME)
STARTTRANSFER_TIME = c.getinfo(c.STARTTRANSFER_TIME)
TOTAL_TIME = c.getinfo(c.TOTAL_TIME)
HTTP_CODE = c.getinfo(c.HTTP_CODE)
SIZE_DOWNLOAD = c.getinfo(c.SIZE_DOWNLOAD)
HEADER_SIZE = c.getinfo(c.HEADER_SIZE)
SPEED_DOWNLOAD=c.getinfo(c.SPEED_DOWNLOAD)

print "HTTP状态码: %s" %(HTTP_CODE)
print "DNS解析时间: %.2f ms" %(NAMELOOKUP_TIME*1000)
print "建立连接时间: %.2f ms" %(CONNECT_TIME*1000)
print "准备传输时间: %.2f ms" %(PRETRANSFER_TIME*1000)
print "传输开始时间: %.2f ms" %(STARTTRANSFER_TIME*1000)
print "传输结束总时间: %.2f ms" %(TOTAL_TIME*1000)
```

```

print "下载数据包大小: %d bytes/s" %(SIZE_DOWNLOAD)
print "HTTP头部大小: %d byte" %(HEADER_SIZE)
print "平均下载速度: %d bytes/s" %(SPEED_DOWNLOAD)

indexfile.close()
c.close()

print time.strftime( ISOTIMEFORMAT, time.gmtime( time.time() ) )
print "=====
```

3. 测试局域网内主机是否 alive 的小脚本

我们在对局域网的网络情况进行维护时，经常会遇到这样的问题，需要收集网络中存活的 IP，这个时候可以写一个 Python 脚本，自动收集某一段的 IP。现在的 IT 技术型公司都比较大，网络工程师一般会规划几个 VLAN（网段），我们可以用如下这个脚本来收集某个 VLAN 下存活的主机，（此脚本在 CentOS 6.4 x86_64 下已测试通过）：

```

#!/usr/bin/python
import os
import re
import time
import sys
import subprocess

lifeline = re.compile(r"(\d) received")
report = ("No response", "Partial Response", "Alive")

print time.ctime()
for host in range(1,254):
    ip = "192.168.1."+str(host)
    pingaling = subprocess.Popen(["ping", "-q", "-c 2", "-r", ip], shell=False,
        stdin=subprocess.PIPE, stdout=subprocess.PIPE)
    print "Testing ",ip,
    while 1:
        pingaling.stdout.flush()
        line = pingaling.stdout.readline()
        if not line: break
        igot = re.findall(lifeline,line)
        if igot:
            print report[int(igot[0])]
    print time.ctime()
```

Python 对空格的要求是非常严谨的，请大家注意下这个问题。脚本虽然短小，但非常实用、精悍，可避免到 Windows 下去下载局域网检测工具。平时我们在日常工作中也应注意多收集、多写一些这样的脚本，以达到简化运维工作的目的。

2.6.3 监控类脚本

在生产环境下，服务器的稳定情况会直接影响公司的生意和信誉，可见其有多重要。

所以，我们需要即时掌握服务器的状态，我们一般会在机房部署 Nagios-Server 作为监控程序，然后用 Shell 和 Python 根据业务需求开发监控插件，实时监控线上业务。

1. Nginx 负载均衡服务器上监控 Nginx 进程脚本

由于笔者公司电子商务业务网站前端的 Load Balance 用到了 Nginx+Keepalived 架构，而 Keepalived 无法进行 Nginx 服务的实时切换，所以设置了一个监控脚本 `nginx_pid.sh`，每隔 5 秒钟就监控一次 Nginx 的运行状态（也可以由 Supervioed 守护进程托管），如果发现有问題就关闭本机的 Keepalived 程序，让 VIP 切换到从 Nginx 负载均衡器上。在对线上环境进行操作的时候，人为地重启主 Master 的 Nginx 机器，从 Nginx 机器在很短的时间内就接管了 VIP 地址，即网站的实际内网地址（此内网地址可以通过防火墙映射为公网 IP），进一步证实了此脚本的有效性，脚本内容如下（此脚本在 CentOS 5.8/6.4 x86_64 下均已测试通过）：

```
#!/bin/bash
while :
do
    nginxpid=`ps -C nginx --no-header | wc -l`
    if [ $nginxpid -eq 0 ];then
        ulimit -SHn 65535
        /usr/local/nginx/sbin/nginx
        sleep 5
        if [ $nginxpid -eq 0 ];then
            /etc/init.d/keepalived stop
        fi
    fi
    sleep 5
done
```

2. 系统文件打开数监测脚本

这个脚本比较方便，可用来查看 Nginx 进程下最大文件打开数，脚本代码如下（此脚本在 CentOS 5.8/6.4 x86_64、Amazon Linux AMI x86_64 下均已测试通过）：

```
#!/bin/bash
for pid in `ps aux |grep nginx |grep -v grep|awk '{print $2}'`
do
    cat /proc/${pid}/limits | grep 'Max open files'
done
```

脚本的运行结果如下所示：

Max open files	65535	65535	files
Max open files	65535	65535	files
Max open files	65535	65535	files
Max open files	65535	65535	files
Max open files	65535	65535	files

3. 监测 MySQL 主从复制是否同步

笔者有不少基于公网类型的网站（没有硬件防火墙，直接置于 IDC 机房）采用的都是 MySQL 主从架构，从机主要起备份数据库和冷备份的作用，虽然从机宕机了问题不大，但也会影响到数据的备份工作；这样的网站有数十个，如果逐个手动检查，每天都要花费不少时间，所以也用了脚本监控。

脚本设计思路：

- 1) 此脚本应该能适应各种各样不同的内外网环境，即 IP 不同的环境。
- 2) 让脚本顺便也监控下 MySQL 是否正常运行。
- 3) Slave 机器的 IO 和 SQL 状态都必须为 YES，缺一不可，这里用到了多重条件判断 -a。

脚本内容如下所示（此脚本在 CentOS 5.8 x86_64 下已测试通过）：

```
#!/bin/bash
#check MySQL_Slave Status
#crontab time 00:10
MYSQLPORT=`netstat -na|grep "LISTEN"|grep "3306"|awk -F[:] '{print $4}'`
MYSQLIP=`ifconfig eth0|grep "inet addr" | awk -F[:] '{print $4}'`
STATUS=$(/usr/local/webserver/mysql/bin/mysql -u yuhongchun -pyuhongchun101 -S /tmp/mysql.sock -e "show slave status\G" | grep -i "running")
IO_env=`echo $STATUS | grep IO | awk '{print $2}'`
SQL_env=`echo $STATUS | grep SQL | awk '{print $2}'`

if [ "$MYSQLPORT" == "3306" ]
then
    echo "mysql is running"
else
    mail -s "warn!server: $MYSQLIP mysql is down" yuhongchun027@163.com
fi

if [ "$IO_env" = "Yes" -a "$SQL_env" = "Yes" ]
then
    echo "Slave is running!"
else
    echo "##### $date #####">> /data/data/check_mysql_slave.log
    echo "Slave is not running!" >> /data/data/check_mysql_slave.log
    mail -s "warn! $MYSQLIP_replicate_error" yuhongchun027@163.com << /data/data/check_mysql_slave.log
fi
```

建议每 10 分钟运行一次，脚本如下：

```
*/10 * * * * root /bin/sh /root/mysql_slave.sh
```

要记得在每台 MySQL 从机上分配一个 yuhongchun 的用户，权限大些也没关系，只限定在本地运行，脚本如下所示：


```
grant all privileges on *.* to "yuhongchun"@"127.0.0.1" identified by "yuhongchun101";
grant all privileges on *.* to "yuhongchun"@"localhost" identified by "yuhongchun101";
```

4. 监控 Python 程序是否正常运行

需求比较简单，主要是监控业务进程 `rsync_redis.py` 是否正常运行，有没有发生 `crash` 的情况。另外，建议将类似于 `rsync_redis.py` 的重要业务进程交由 `Supervised` 守护进程托管。此脚本内容如下所示（此脚本在 Amazon Linux AMI x86_64 下已测试通过）：

```
#!/bin/bash
sync_redis_status=`ps aux | grep sync_redis.py | grep -v grep | wc -l`
if [ ${sync_redis_status} != 1 ]; then
    echo "Critical! sync_redis is Died"
    exit 2
else
    echo "OK! sync_redis is Alive"
    exit 0
fi
```

2.6.4 开发类脚本

业务需求在不断地变化，有时候互联网上的开源方案并不能全部解决，这个时候就需要自己写一些开发类的脚本来满足工作中的需求了，虽然很多时候脚本都可以独立运行，但笔者的做法还是尽量将其 `return` 结果写成 Nagios 能够识别的格式，以便配合 Nagios 发送报警邮件和信息。

1. 监测 redis 是否正常运行

笔者接触的线上 NoSQL 业务主要是 redis 数据库，多用于处理大量数据的高访问负载需求。为了最大化地利用资源，每个 redis 实例分配的内存并不是很大，有时候程序组同事导入数据量大的 IP list 时会导致 redis 实例崩溃，所以笔者开发了一个 redis 监测脚本并配合 Nagios 进行工作，脚本内容如下所示（此脚本在 Amazon Linux AMI x86_64 下已测试通过）：

```
#!/usr/bin/python
#Check redis Nagios Plungin, Please install the redis-py module.
import redis
import sys

STATUS_OK = 0
STATUS_WARNING = 1
STATUS_CRITICAL = 2

HOST = sys.argv[1]
PORT = int(sys.argv[2])
WARNING = float(sys.argv[3])
CRITICAL = float(sys.argv[4])
```

```

def connect_redis(host, port):
    r = redis.Redis(host, port, socket_timeout = 5, socket_connect_timeout = 5)
    return r

def main():
    r = connect_redis(HOST, PORT)
    try:
        r.ping()
    except:
        print HOST,PORT,'down'
        sys.exit(STATUS_CRITICAL)

    redis_info = r.info()
    used_mem = redis_info['used_memory']/1024/1024/1024.0
    used_mem_human = redis_info['used_memory_human']

    if WARNING <= used_mem < CRITICAL:
        print HOST,PORT,'use memory warning',used_mem_human
        sys.exit(STATUS_WARNING)
    elif used_mem >= CRITICAL:
        print HOST,PORT,'use memory critical',used_mem_human
        sys.exit(STATUS_CRITICAL)
    else:
        print HOST,PORT,'use memory ok',used_mem_human
        sys.exit(STATUS_OK)

if __name__ == '__main__':
    main()

```

2. 监测机器的 IP 连接数

需求其实比较简单，先统计 IP 连接数，如果 ip_conns 值小于 15 000 则显示为正常，介于 15 000 至 20 000 之间为警告，如果超过 20 000 则报警，脚本内容如下所示（此脚本在 Amazon Linux AMI x86_64 下已测试通过）：

```

#!/bin/bash
#Nagios plugin For ip connects
# $1 = 15000 $2 = 20000
ip_conns=`netstat -an | grep tcp | grep EST | wc -l`
messages=`netstat -ant | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]]' | tr
-s '\n' ' ' | sed -r 's/(.*)/\1\n/g'`

if [ $ip_conns -lt $1 ]
then
    echo "$messages,OK -connect counts is $ip_conns"
    exit 0
fi
if [ $ip_conns -gt $1 -a $ip_conns -lt $2 ]
then
    echo "$messages,Warning -connect counts is $ip_conns"

```

```

        exit 1
    fi
    if [ $ip_conns -gt $2 ]
    then
        echo "$messages,Critical -connect counts is $ip_conns"
        exit 2
    fi

```

3. 监测机器的 CPU 利用率脚本

线上的 bidder 业务机器，在业务繁忙的高峰期会出现 CPU 利用率达到 100% (sys%+user%)，导致后面的流量打在上面却完全进不去的情况，但此时机器、系统负载及 Nginx+Lua 进程都是完全正常的，所以这种情况下需要开发一个 CPU 利用率脚本，在超过自定义阈值时报警，方便运维人员批量添加 bidder AMI 机器以应对峰值，AWS EC2 实例机器是可以以小时来计费的，大家在这里也要注意分清系统负载和 CPU 利用率之间的区别。脚本内容如下所示（此脚本在 Amazon Linux AMI x86_64 下已测试通过）：

```

#!/bin/bash
# =====
# CPU Utilization Statistics plugin for Nagios
#
# USAGE          :      ./check_cpu_utili.sh [-w <user,system,iowait>] [-c
#                   <user,system,iowait>] ( [ -i <intervals in second> ] [ -n <report number> ] )
#
# # Example: ./check_cpu_utili.sh
# #           ./check_cpu_utili.sh -w 70,40,30 -c 90,60,40
# #           ./check_cpu_utili.sh -w 70,40,30 -c 90,60,40 -i 3 -n 5
# -----
# # Paths to commands used in this script.  These may have to be modified to match
# # your system setup.
# IOSTAT="/usr/bin/iostat"
#
# # Nagios return codes
# STATE_OK=0
# STATE_WARNING=1
# STATE_CRITICAL=2
# STATE_UNKNOWN=3
#
# # Plugin parameters value if not define
# LIST_WARNING_THRESHOLD="70,40,30"
# LIST_CRITICAL_THRESHOLD="90,60,40"
# INTERVAL_SEC=1
# NUM_REPORT=1
#
# # Plugin variable description
# PROGNAME=$(basename $0)
#
if [ ! -x $IOSTAT ]; then
    echo "UNKNOWN: iostat not found or is not executable by the nagios user."
    exit $STATE_UNKNOWN

```

```

fi

print_usage() {
    echo ""
    echo "$PROGNAME $RELEASE - CPU Utilization check script for Nagios"
    echo ""
    echo "Usage: check_cpu_utili.sh -w -c (-i -n)"
    echo ""
    echo "    -w Warning threshold in % for warn_user, warn_system, warn_iowait
CPU (default : 70,40,30)"
    echo "    Exit with WARNING status if cpu exceeds warn_n"
    echo "    -c Critical threshold in % for crit_user, crit_system, crit_iowait
CPU (default : 90,60,40)"
    echo "    Exit with CRITICAL status if cpu exceeds crit_n"
    echo "    -i Interval in seconds for iostat (default : 1)"
    echo "    -n Number report for iostat (default : 3)"
    echo "    -h Show this page"
    echo ""
    echo "Usage: $PROGNAME"
    echo "Usage: $PROGNAME --help"
    echo ""
    exit 0
}

print_help() {
    print_usage
    echo ""
    echo "This plugin will check cpu utilization (user,system,CPU_Iowait in %)"
    echo ""
    exit 0
}

# Parse parameters
while [ $# -gt 0 ]; do
    case "$1" in
        -h | --help)
            print_help
            exit $STATE_OK
            ;;
        -v | --version)
            print_release
            exit $STATE_OK
            ;;
        -w | --warning)
            shift
            LIST_WARNING_THRESHOLD=$1
            ;;
        -c | --critical)
            shift
            LIST_CRITICAL_THRESHOLD=$1
            ;;
    esac
done

```



```

-i | --interval)
    shift
    INTERVAL_SEC=$1
    ;;
-n | --number)
    shift
    NUM_REPORT=$1
    ;;
*) echo "Unknown argument: $1"
   print_usage
   exit $STATE_UNKNOWN
   ;;
esac

shift
done

# List to Table for warning threshold (compatibility with
TAB_WARNING_THRESHOLD=(`echo $LIST_WARNING_THRESHOLD | sed 's/,/ /g'`)
if [ "${#TAB_WARNING_THRESHOLD[@]}" -ne "3" ]; then
    echo "ERROR : Bad count parameter in Warning Threshold"
    exit $STATE_WARNING
else
    USER_WARNING_THRESHOLD=`echo ${TAB_WARNING_THRESHOLD[0]}`
    SYSTEM_WARNING_THRESHOLD=`echo ${TAB_WARNING_THRESHOLD[1]}`
    IOWAIT_WARNING_THRESHOLD=`echo ${TAB_WARNING_THRESHOLD[2]}`
fi

# List to Table for critical threshold
TAB_CRITICAL_THRESHOLD=(`echo $LIST_CRITICAL_THRESHOLD | sed 's/,/ /g'`)
if [ "${#TAB_CRITICAL_THRESHOLD[@]}" -ne "3" ]; then
    echo "ERROR : Bad count parameter in CRITICAL Threshold"
    exit $STATE_WARNING
else
    USER_CRITICAL_THRESHOLD=`echo ${TAB_CRITICAL_THRESHOLD[0]}`
    SYSTEM_CRITICAL_THRESHOLD=`echo ${TAB_CRITICAL_THRESHOLD[1]}`
    IOWAIT_CRITICAL_THRESHOLD=`echo ${TAB_CRITICAL_THRESHOLD[2]}`
fi

if [ ${TAB_WARNING_THRESHOLD[0]} -ge ${TAB_CRITICAL_THRESHOLD[0]} -o ${TAB_WARNING_THRESHOLD[1]} -ge ${TAB_CRITICAL_THRESHOLD[1]} -o ${TAB_WARNING_THRESHOLD[2]} -ge ${TAB_CRITICAL_THRESHOLD[2]} ]; then
    echo "ERROR : Critical CPU Threshold lower as Warning CPU Threshold "
    exit $STATE_WARNING
fi

CPU_REPORT=`iostat -c $INTERVAL_SEC $NUM_REPORT | sed -e 's/,/./g' | tr -s ' ' ';' | sed '/^$/d' | tail -1`
CPU_REPORT_SECTIONS=`echo ${CPU_REPORT} | grep ';' -o | wc -l`
CPU_USER=`echo $CPU_REPORT | cut -d ";" -f 2`
CPU_SYSTEM=`echo $CPU_REPORT | cut -d ";" -f 4`
CPU_IOWAIT=`echo $CPU_REPORT | cut -d ";" -f 5`

```

```

CPU_STEAL=`echo $CPU_REPORT | cut -d ";" -f 6`
CPU_IDLE=`echo $CPU_REPORT | cut -d ";" -f 7`
NAGIOS_STATUS="user=${CPU_USER}%,system=${CPU_SYSTEM}%,iowait=${CPU_IOWAIT}%,idle=${CPU_IDLE}%"
NAGIOS_DATA="CpuUser=${CPU_USER}%;${TAB_WARNING_THRESHOLD[0]};${TAB_CRITICAL_THRESHOLD[0]};0"

CPU_USER_MAJOR=`echo $CPU_USER | cut -d "." -f 1`
CPU_SYSTEM_MAJOR=`echo $CPU_SYSTEM | cut -d "." -f 1`
CPU_IOWAIT_MAJOR=`echo $CPU_IOWAIT | cut -d "." -f 1`
CPU_IDLE_MAJOR=`echo $CPU_IDLE | cut -d "." -f 1`

# Return
if [ ${CPU_USER_MAJOR} -ge $USER_CRITICAL_THRESHOLD ]; then
    echo "CPU STATISTICS OK:${NAGIOS_STATUS} | CPU_USER=${CPU_USER}%;70;90;0;100"
    exit $STATE_CRITICAL
elif [ ${CPU_SYSTEM_MAJOR} -ge $SYSTEM_CRITICAL_THRESHOLD ]; then
    echo "CPU STATISTICS OK:${NAGIOS_STATUS} | CPU_USER=${CPU_USER}%;70;90;0;100"
    exit $STATE_CRITICAL
elif [ ${CPU_IOWAIT_MAJOR} -ge $IOWAIT_CRITICAL_THRESHOLD ]; then
    echo "CPU STATISTICS OK:${NAGIOS_STATUS} | CPU_USER=${CPU_USER}%;70;90;0;100"
    exit $STATE_CRITICAL
elif [ ${CPU_USER_MAJOR} -ge $USER_WARNING_THRESHOLD ] && [ ${CPU_USER_MAJOR} -lt $USER_CRITICAL_THRESHOLD ]; then
    echo "CPU STATISTICS OK:${NAGIOS_STATUS} | CPU_USER=${CPU_USER}%;70;90;0;100"
    exit $STATE_WARNING
elif [ ${CPU_SYSTEM_MAJOR} -ge $SYSTEM_WARNING_THRESHOLD ] && [ ${CPU_SYSTEM_MAJOR} -lt $SYSTEM_CRITICAL_THRESHOLD ]; then
    echo "CPU STATISTICS OK:${NAGIOS_STATUS} | CPU_USER=${CPU_USER}%;70;90;0;100"
    exit $STATE_WARNING
elif [ ${CPU_IOWAIT_MAJOR} -ge $IOWAIT_WARNING_THRESHOLD ] && [ ${CPU_IOWAIT_MAJOR} -lt $IOWAIT_CRITICAL_THRESHOLD ]; then
    echo "CPU STATISTICS OK:${NAGIOS_STATUS} | CPU_USER=${CPU_USER}%;70;90;0;100"
    exit $STATE_WARNING
else
    echo "CPU STATISTICS OK:${NAGIOS_STATUS} | CPU_USER=${CPU_USER}%;70;90;0;100"
    exit $STATE_OK
fi

```

此脚本参考了 Nagios 的官方文档 <https://exchange.nagios.org/> 并进行了代码精简和移植, 源代码是运行在 `ksh` 下面的, 这里将其移植到了 `bash` 下面, `ksh` 下定义数组的方式跟 `bash` 还是有区别的; 另外有一点也请大家注意, Shell 本身是不支持浮点运算的, 但可以通过 `bc` 或 `awk` 的方式来处理。

另外, 若要配合 PNP4nagios 出图 (PNP4nagios 可以观察一段周期内的 CPU 利用率峰值), 此脚本还可以更精简, 脚本内容如下所示 (此脚本在 Amazon Linux AMI x86_64 下已测试通过):

```
#!/bin/bash
# Nagios return codes
STATE_OK=0
STATE_WARNING=1
STATE_CRITICAL=2
STATE_UNKNOWN=3

# Plugin parameters value if not define
LIST_WARNING_THRESHOLD="90"
LIST_CRITICAL_THRESHOLD="95"
INTERVAL_SEC=1
NUM_REPORT=5

CPU_REPORT=`iostat -c $INTERVAL $NUM_REPORT | sed -e 's/,./g' | tr -s ' ' ';' | sed
'/^$/d' | tail -1`
CPU_REPORT_SECTIONS=`echo ${CPU_REPORT} | grep ';' -o | wc -l`
CPU_USER=`echo $CPU_REPORT | cut -d ";" -f 2`
CPU_SYSTEM=`echo $CPU_REPORT | cut -d ";" -f 4`
# Add for integer shell issue
CPU_USER_MAJOR=`echo $CPU_USER | cut -d "." -f 1`
CPU_SYSTEM_MAJOR=`echo $CPU_SYSTEM | cut -d "." -f 1`
CPU_UTILI_COU=`echo ${CPU_USER} + ${CPU_SYSTEM}|bc`
CPU_UTILI_COUNTER=`echo $CPU_UTILI_COU | cut -d "." -f 1`

# Return
if [ ${CPU_UTILI_COUNTER} -lt ${LIST_WARNING_THRESHOLD} ]
then
    echo "OK - CPUCOU=${CPU_UTILI_COU}% | CPUCOU=${CPU_UTILI_COUNTER}%;80;90"
    exit ${STATE_OK}
fi
if [ ${CPU_UTILI_COUNTER} -gt ${LIST_WARNING_THRESHOLD} -a ${CPU_UTILI_COUNTER}
-lt ${LIST_CRITICAL_THRESHOLD} ]
then
    echo "Warning - CPUCOU=${CPU_UTILI_COUNTER}% | CPUCOU=${CPU_UTILI_COUNTER}%;80;90"
    exit ${STATE_WARNING}
fi
if [ ${CPU_UTILI_COUNTER} -gt ${LIST_CRITICAL_THRESHOLD} ]
then
    echo "Critical - CPUCOU=${CPU_UTILI_COUNTER}% | CPUCOU=${CPU_UTILI_COUNTER}%;80;90"
    exit ${STATE_CRITICAL}
fi
```

2.6.5 自动化类脚本

1. 批量生成账户脚本

在内网开发环境中,有时需要为开发组的同事批量生成账户,如果手动添加的话会非常麻烦,这时可以写一段 Shell 脚本来自动完成这项工作。在首次登录时密码均是统一的,在移交给开发人员使用时让他们自行更改即可,脚本代码如下(此脚本在 CentOS 5.8 / 6.4 x86_64 下均已测试通过):

```
#!/bin/bash
#此脚本应用于开发环境下批量生成用户
for name in tom jerry joe jane yhc brain
do
    useradd $name
    echo redhat | passwd --stdin $name
done
```

passwd --stdin 这行代码的作用是将前面的输入通过管道命令作为自己的输出，从而避免脚本交互，达到自动化的目的。

笔者个人觉得用脚本的方式来批量自动添加用户的方法较之 Ansible 的 user 模块更为简便，有兴趣的朋友也可以研究比较下。

2. 系统初始化脚本

此脚本用于新装 Linux 的相关配置工作，比如禁用 iptables、SELinux 及 ipv6，优化系统内核，停掉一些没必要启动的系统服务等。此脚本可用于公司内部的开发机器的批量部署，脚本代码如下所示（此脚本在 CentOS 6.4 x86_64 下已测试通过）：

```
#!/bin/bash
#添加epel外部yum扩展源
cd /usr/local/src
wget http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
rpm -ivh epel-release-6-8.noarch.rpm
#安装gcc基础库文件及sysstat工具
yum -y install gcc gcc-c++ vim-enhanced unzip unrar sysstat
#配置ntpddate自动对时
yum -y install ntp
echo "01 01 * * * /usr/sbin/ntpdate ntp.api.bz    >> /dev/null 2>&1" >> /etc/crontab
ntpdate ntp.api.bz
service crond restart
#配置文件的ulimit值
ulimit -SHn 65534
echo "ulimit -SHn 65534" >> /etc/rc.local
cat >> /etc/security/limits.conf << EOF
*                soft          nofile          65534
*                hard          nofile          65534
EOF
```

#基础系统内核优化

```
cat >> /etc/sysctl.conf << EOF
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_syn_retries = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_fin_timeout = 1
net.ipv4.tcp_keepalive_time = 1200
net.ipv4.ip_local_port_range = 1024 65535
net.ipv4.tcp_max_syn_backlog = 16384
net.ipv4.tcp_max_tw_buckets = 36000
```



```

net.ipv4.route.gc_timeout = 100
net.ipv4.tcp_syn_retries = 1
net.ipv4.tcp_synack_retries = 1
net.core.somaxconn = 16384
net.core.netdev_max_backlog = 16384
net.ipv4.tcp_max_orphans = 16384
EOF
/sbin/sysctl -p

#禁用control-alt-delete组合键以防止误操作
sed -i 's@ca::ctrlaltdel:/sbin/shutdown -t3 -r now@#ca::ctrlaltdel:/sbin/shutdown
-t3 -r now@' /etc/inittab
#关闭SELinux
sed -i 's@SELINUX=enforcing@SELINUX=disabled@' /etc/selinux/config
#关闭iptables
service iptables stop
chkconfig iptables off
#ssh服务配置优化,请保持机器中至少有一个具有sudo权限的用户,下面的配置会禁止root远程登录
sed -i 's@#PermitRootLogin yes@PermitRootLogin no@' /etc/ssh/sshd_config #禁止
root远程登录
sed -i 's@#PermitEmptyPasswords no@PermitEmptyPasswords no@' /etc/ssh/sshd_config
#禁止空密码登录
sed -i 's@#UseDNS yes@UseDNS no@' /etc/ssh/sshd_config /etc/ssh/sshd_config
service sshd restart
#禁用ipv6地址
echo "alias net-pf-10 off" >> /etc/modprobe.d/dist.conf
echo "alias ipv6 off" >> /etc/modprobe.d/dist.conf
chkconfig ip6tables off
#vim基础语法优化
echo "syntax on" >> /root/.vimrc
echo "set nohlsearch" >> /root/.vimrc
#精简开机自启动服务,安装最小化服务的机器初始可以只保留crond、network、rsyslog、sshd这4个服务。
for i in `chkconfig --list|grep 3:on|awk '{print $1}`;do chkconfig --level 3 $i off;done
for CURSRV in crond rsyslog sshd network;do chkconfig --level 3 $CURSRV on;done
#重启服务器
reboot

```

2.7 小结

本章向大家详细说明了 Shell 的基础语法,以及 sed 和 awk 在日常工作中的使用案例,并用 Shell 命令 grep 和 find 结合正则表达式演示了正则表达式的一些基础用法。在后面的实例中,又根据备份类、监控类、统计类、自动化运维类、运维开发类向大家演示了在生产环境下我们经常用到的 Shell 和 Python 脚本。我们在感叹 Shell 脚本强大的管理功能的同时,也应该比较清楚 Shell 脚本在开发功能上的不足,而 Python 正好能够弥补这个缺点,它继承了传统编译语言的强大性和通用性,同时也借鉴了简单脚本和解释语言的易用性,运行速度也不慢,适合网站开发,正好可以弥补 Shell 脚本的不足。结合这两种脚本语言,我们的系统运维工作和 DevOps 工作会更加得心应手。

轻量级自动化运维工具 Fabric 详解

近期公司的业务系统代码发布频繁，笔者同时在几个项目组里面穿插工作，发现发布和运维的工作都相当机械，加上频率比较高，导致时间的浪费也比较多。很多测试工作，例如通过 SSH 登录到测试环境，推送代码，然后修改 Bug 进行测试，这些操作都是非常机械并且具有重复性的。更让人郁闷的是，每次的操作都是相同的，命令基本上都是一样的，并且是在多台机器上执行，很难在本机上以一个脚本来搞定，主要时间都浪费在使用 SSH 登录和输入命令上了。这个时候需要一个轻量级的自动化运维工具，来帮助我们解决这些问题，Fabric 就顺应这个需求而出现了，它非常适合于这些简单的、重复性的远程操作。Fabric 是基于 Python 语言开发的，前文 2.1 节就提到过，Python 应用非常火爆，接下来看看 Python 的应用领域及其流行的原因。

3.1 Python 语言的应用领域

1. 云计算基础设施

云计算平台分为私有云和公有云。私有云平台如大名鼎鼎的 OpenStack，就是以 Python 语言编写的。公有云，无论是 AWS、Azure、GCE（Google Compute Engine）、阿里云还是青云，都提供了 Python SDK，其中 GCE 只提供了 Python 和 JavaScript 的 SDK，青云只提供了 Python SDK。由此可见各家云平台对 Python 的重视。



注意 软件开发工具包（Software Development Kit, SDK）一般是一些开发工具的集合，用于为特定的软件包、软件框架、硬件平台、操作系统等创建应用软件。

2. DevOps

DevOps, 中文名译作开发型运维。在互联网时代, 只有能够快速试验新想法, 并在第一时间, 安全、可靠地交付业务价值, 才能保持竞争力。DevOps 推崇的自动化构建、测试、部署及系统度量等技术实践, 在互联网时代是尤其重要的。

自动化构建是因应用而异的, 如果是 Python 应用, 因为有 `setuptools`、`pip`、`virtualenv`、`tox`、`flake8` 等工具的存在, 所以自动化构建非常简单。而且, 因为几乎所有的 Linux 版本都内置了 Python 解释器, 所以用 Python 做自动化, 系统不需要预安装什么软件。

自动化测试方面, 目前流行的自动化测试框架有 Robot Framework、Cucumber、Lettuce 三种。基于 Python 的 Robot Framework 是企业级应用最喜欢的自动化测试框架, 而且和语言无关。Cucumber 也有很多支持者。基于 Python 的 Lettuce 可以实现完全一样的功能。此外, Locust (一个基于 Python 开发的开源负载测试工具) 也开始在自动化性能测试方面受到越来越多的关注。

自动化配置管理工具, 老牌的如 Chef 和 Puppet, 是基于 Ruby 语言开发设计的, 目前仍保持着强劲的势头。不过, 新生代 Ansible、SaltStack, 以及轻量级的自动化运维工具 Fabric, 均为 Python 语言开发。由于它们较前两者的设计更为轻量化, 因此受到越来越多开发者的欢迎, 并且已经给 Chef 和 Puppet 制造了不少的竞争压力。

3. 网络爬虫

大数据的数据从哪里来? 除了部分企业有能力自己产生大量的数据, 大部分时候, 是需要依靠网络爬虫来抓取互联网数据进行分析的。

网络爬虫是 Python 的传统强势领域, 最流行的爬虫框架 Scrapy、HTTP 工具包 `urllib2`、HTML 解析工具 `Beautiful Soup`、XML 解析器 `lxml` 等, 都是能够独当一面的类库。笔者公司的分布式网络爬虫程序也是基于 Scrapy 开发的。不过, 网络爬虫并不仅仅是打开网页, 解析 HTML 这么简单。高效的爬虫要能够支持大量灵活的并发操作, 常常要能够同时抓取几千甚至上万个网页, 使用传统的线程池方式资源浪费比较大, 线程数上千之后系统资源基本上就全浪费在线程调度上了。由于 Python 能够很好地支持协程 (Coroutine) 操作, 因此基于 Python 发展了很多并发库, 如 `Gevent`、`Eventlet`, 还有 `Celery` 之类的分布式任务框架等。被认为是比 AMQP 更高效的 ZeroMQ 最早提供的也是 Python 版本。有了对高并发的支持, 网络爬虫才可以真正达到大数据规模。

4. 数据处理

从统计理论, 到数据挖掘、机器学习, 再到最近几年提出来的深度学习理论, 数据科学正处于百花齐放的时代。数据科学家们都用什么语言编程呢? Python 是数据科学家最喜欢的语言之一。和 R 语言不同, Python 本身就是一门工程性语言, 数据科学家用 Python 实现的算法, 可以直接用在产品中, 这对于初创的大数据公司来说, 是非常有利于节省成本的。正是基于数据科学家对 Python 和 R 的热爱, Spark 为了“讨好”数据科学家, 对这两

种语言都提供了非常好的支持。

Python 的数据处理相关类库非常多。比如，高性能的科学计算类库 NumPy 和 SciPy，给其他高级算法打下了非常好的基础；Matplotlib 让 Python 画图变得像 Matlab 一样简单；Scikit-learn 和 Milk 实现了很多机器学习算法，基于这两个库实现的 Pylearn2，是深度学习领域的重要成员；Theano 利用 GPU 加速，实现了高性能数学符号计算和 multidimensional 矩阵计算。当然，还有 Pandas，一个在工程领域已被广泛使用的大数据处理类库，其 DataFrame 的设计借鉴自 R 语言，后来又启发 Spark 项目实现了类似机制。

除了这些领域以外，Python 还被广泛应用于 Web 开发、游戏开发、手机开发、数据库开发等众多领域。

3.2 选择 Python 的原因

对于开发工程师而言，Python 的优雅和简洁无疑具有最大的吸引力，在 Python 交互式环境中，执行 `import this` 命令，读一读 Python 之禅，你就会明白 Python 为什么如此吸引人了。Python 社区一直非常有活力，和 NodeJS 社区软件包的爆炸式增长不同，Python 的软件包增长速度一直比较稳定，同时软件包的质量也相对较高。有很多人诟病 Python 对于空格的要求过于苛刻，但正是基于这个严格的要求，才使得 Python 在做大型项目时比其他语言更有优势。OpenStack 项目的代码总共超过 200 万行，也证明了这一点。

对于运维工程师而言，Python 的最大优势在于，几乎所有的 Linux 发行版本都内置了 Python 解释器。Shell 虽然功能强大，但缺点很多：语法不够优雅，不支持面向对象、没有第三方库支持，所以在写比较复杂的任务时会很痛苦。用 Python 替代 Shell，完成一些 Shell 实现不了的复杂任务，对于运维人员、运维工程师来说，是一次解放。

对于 DevOps 而言，Python 的优势在于它是一门强大的“胶水语言”，特别适合应用于 Web 后端、服务器开发，其优点如下：

- Python 的代码风格简洁易懂、易于维护，包括语法优势不用写大括号，代码注释风格统一，强调做一个事情只有一种方法等。
- 有着丰富的 Web 开源框架，主流的包括 Web2py、Web.py、Zope2、Pyramid、Django 等。
- 具有跨平台能力，支持 Mac、Linux、Windows 等系统。
- Python 可用库和模块比较多，非常方便。
- Python 社区非常活跃，在其社区里基本上能够找到一切你所需要的答案。

基于以上原因，我们还有什么理由不选择 Python 呢？

3.3 Python 的版本说明

关于 Python 的版本需要重点说明下，Python 的 2.x 版本和 3.x 版本的差异还是很大的，

语法上也有很多是完全不一样的，这里以线上环境说明。在线上环境中，暂时还是只用 Python 2.7 版本，具体原因如下：

- ❑ 由于历史原因，笔者公司业务系统的 Python 代码是基于 Python 2.7 版本开发的，如果向 Python3.x 版本移植的话工作量太大，而且不能保证系统的稳定性，所以暂时不考虑采用 Python3.x 版本。
- ❑ 现在采用的很多第三方类库都只提供了 Python2.x 版本，而没有提供 Python3.x 版本。
- ❑ 开发环境为了跟线上环境保持一致，也主要是 Python 2.7 版本。

基于上面的原因，本章的内容也以 Python 2.7 版本为主，下面所有有关 Python 的代码都是基于 Python 2.7 版本的，并没有涉及 Python 3.x 版本，这一点希望大家注意。

3.4 增强的交互式环境 IPython

虽然 Python 自带了原生的 Python Shell，但功能上还是比 IPython 略逊一筹。IPython 是一种基于 Python 的交互式解释器。相较于原生的 Python Shell，IPython 提供了更为强大的编辑和交互功能。IPython 拥有一套复杂的并行和分配计算结构，使得各种并行应用能够交互式地被开发、执行、调试和监控。事实上，IPython 中的“I”就代表“交互”。这个解释器的强大使其不仅可以作为 Python 的解释器，甚至还可以直接作为系统管理员的工作环境。IPython 具有以下特征。

- ❑ Tab 补全：可以有效地补齐 Python 语言的模块、方法和类等（原生的 Python Shell 不支持 Tab 补全功能）。
- ❑ magic 函数：内置了很多函数用来实现各种特征。
- ❑ 宏：可以将一段代码定义为一个宏，以便日后运行。
- ❑ 历史记录：提供了强大的历史记录功能。
- ❑ 执行系统命令：可以直接在交互式 Shell 中执行系统命令。
- ❑ 社区支持：IPython 有着非常活跃的社区支持。

下面介绍 IPython 的安装过程。

IPython 的主页是 <http://ipython.scipy.org/>，其中有关于 IPython 的官方资源，包括文档、下载和常见问题等。在 CentOS 6.4 x86_64 上通过 yum 安装 IPython 是非常简单的，步骤如下：

1) 下载 epel 并安装 epel 源，命令如下所示：

```
wget http://ftp.linux.ncsu.edu/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
rpm -ivh epel-release-6-8.noarch.rpm
```

2) 通过 yum 安装 IPython，命令如下所示：

```
yum -y install ipython
```

安装成功以后，就可以直接输入命令 `ipython` 来启动 IPython 解释器了，大家可以对比

一下, IPython 界面比原生的 Python Shell 界面更漂亮, 如图 3-1 所示。

```
[root@localhost yum.repos.d]# ipython
Python 2.6.6 (r266:84292, Jul 23 2015, 15:22:56)
Type "copyright", "credits" or "license" for more information.

IPython 0.13.2 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref  -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.

In [1]:
```

图 3-1 IPython 安装成功界面

3.5 Python(x,y) 介绍

Python(x,y) 是 Windows 下一个免费的科学和工程开发包, 提供数学计算、数据分析和可视化展示。从名字就能看出来这个发行版附带了科学计算方面的很多常用库, 另外还有用于桌面软件界面制作的 PyQt, 以及进行文档处理、生成 EXE 文件等的常用库。此外, 它还包含了大量的工具, 如 IDE、制图制表的工具、加强的互动 Shell 等。下文提到的很多软件在此发行版中都有附带。在其他方面, Python(x, y) 还附带了手工整理出的所有库的离线文档, 每个小版本升级都提供了单独的补丁。

Python(x,y) 安装成功以后, 就默认自带了 IPython、PyDev (Python IDE) 这些软件包, 非常方便, 推荐大家在 Windows 下安装此软件包来学习 Python 的基础语法。Python(x,y) 里面包含的软件如图 3-2 所示。

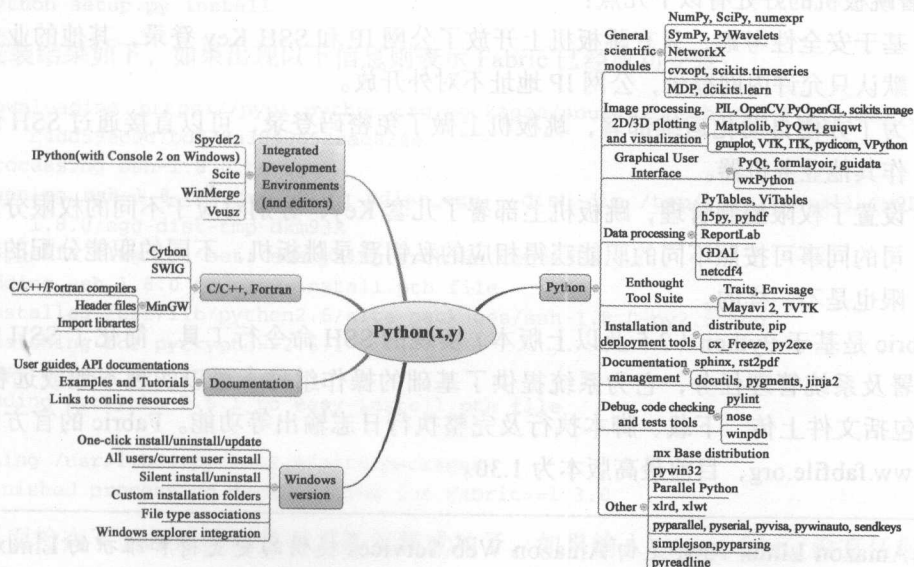


图 3-2 Python(x,y) 包含的软件图示

3.6 轻量级自动化运维工具 Fabric 介绍

笔者公司目前的数据中心采用的是分布式部署方案，在全球多地都有数据中心。数据中心采用的是 AWS EC2 机器，在核心的数据中心里，EC2 机器的数量比较多，基本上每个数据中心都在运行着几百台 AWS EC2 机器，而且业务繁忙的时候，会通过 AWS AMI (Amazon 系统映像) 直接上线几十台相同业务的 EC2 机器，它们的机器类型、系统应用和配置文件基本上都是一模一样的，很多时候需要修改相同的配置文件和执行相同的操作，这个时候为了避免重复性的劳动就需要用到自动化运维工具，轻量级自动化运维工具 Fabric 在这里是首选。Fabric 是基于 Python 语言开发的，是开发组同事的最爱。为了方便自动化运维，我们在每个数据中心都部署了跳板机（在跳板机上部署了 Fabric），其物理拓扑图如图 3-3 所示。



图 3-3 跳板机物理拓扑图

部署跳板机的好处有以下几点：

- ❑ 基于安全性考虑，只有跳板机上开放了公网 IP 和 SSH Key 登录，其他的业务机器默认只允许内网登录，公网 IP 地址不对外开放。
- ❑ 为了方便自动化运维部署，跳板机上做了免密码登录，可以直接通过 SSH 命令操作其他业务机器。
- ❑ 设置了权限控制管理，跳板机上部署了几套 Key，分别对应于不同的权限分配，公司的同事可按照不同的职能获得相应的私钥登录跳板机，不同的职能分配的相应权限也是不一样的。

Fabric 是基于 Python (2.5 及以上版本) 实现的 SSH 命令行工具，简化了 SSH 的应用程序部署及系统管理任务，它为系统提供了基础的操作组件，可以实现本地或远程 Shell 命令，包括文件上传、下载、脚本执行及完整执行日志输出等功能。Fabric 的官方地址为 <http://www.fabfile.org>，目前最高版本为 1.30。



提示

Amazon Linux AMI 是由 Amazon Web Services 提供的受支持和维护的 Linux 映像，用于 Amazon Elastic Compute Cloud (Amazon EC2)。旨在为 Amazon EC2 上运行

的应用程序提供稳定、安全和高性能的执行环境。它支持最新的 EC2 实例类型功能，并包括可与 AWS 轻松集成的软件包。Amazon Web Services 为运行 Amazon Linux AMI 的所有实例提供了持续的安全性和维护更新。Amazon Linux AMI 对于 Amazon EC2 用户是免费的。

3.6.1 Fabric 的安装

安装 Fabric 时，可以选择采用 Python 的 pip、easy_install 及源码安装方式，这些方式能够很方便地解决包依赖关系。大家可以根据系统环境自行选择最优的安装方法，如果选择 pip 或 easy_install 安装方式，则其安装命令如下（如果系统是最小化安装，记得先提前安装好 gcc、gcc-c++、make 这些基础开发包和 python-pip）：

```
yum -y install make gcc gcc++ python-devel python-pip
```

pip 是安装 Python 包的工具，提供了安装包、列出已经安装的包、升级包及卸载包的功能，可以通过 pip 工具直接安装 Fabric，命令如下：

```
pip install fabric
```

这里推荐源码安装，安装步骤如下所示：

```
yum -y install python-setuptools
cd /usr/local/src
wget https://pypi.python.org/packages/source/F/Fabric/Fabric-1.3.0.tar.gz --no-check-certificate
tar xvf Fabric-1.3.0.tar.gz
cd Fabric-1.3.0
python setup.py install
```

安装结果如下，如果出现以下信息则表示 Fabric 已经成功安装：

```
Downloading https://pypi.python.org/packages/source/s/ssh/ssh-1.8.0.tar.gz#md5=b
c4dd59ec0c7bdf78a3840652cac824e
Processing ssh-1.8.0.tar.gz
Running ssh-1.8.0/setup.py -q bdist_egg --dist-dir /tmp/easy_install-Cw9Pkj/ssh-
1.8.0/egg-dist-tmp-dkm93k
zip_safe flag not set; analyzing archive contents...
Adding ssh 1.8.0 to easy-install.pth file
Installed /usr/lib/python2.6/site-packages/ssh-1.8.0-py2.6.egg
Searching for pycrypto==2.6.1
Best match: pycrypto 2.6.1
Adding pycrypto 2.6.1 to easy-install.pth file

Using /usr/lib64/python2.6/site-packages
Finished processing dependencies for Fabric==1.3.0
```

下面检查下 Fabric 模块是否正常安装成功了，如果输入 `import fabric` 没有任何错误提示则表示已经成功安装，命令如下所示：

```
Python 2.6.6 (r266:84292, Jul 23 2015, 15:22:56)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-11)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import fabric
>>>
```

3.6.2 命令行入口 fab 命令详解

fab 作为 Fabric 的命令行入口，提供了丰富的参数调用，命令格式如下：

```
fab [options] -- [shell command]
```

其中，

- ❑ -l: 显示定义好的任务函数名。
- ❑ -f: 指定 fab 入口文件，默认入口文件名为 fabfile.py，如果当前目录不存在 fabfile.py，则必须用 -f 参数指定一个新的文件，否则会报错。
- ❑ -g: 指定网关设备，比如跳板机环境，填写跳板机 IP 即可。
- ❑ -H: 指定目标主机，多台主机用 “,” 号分隔。
- ❑ -P: 以异步并行方式运行多个主机任务，默认为串行运行。
- ❑ -R: 指定角色 (role)，以角色名区分不同的业务组设备。
- ❑ -t: 设置设备连接超时时间。
- ❑ -T: 设置远程主机命令执行超时时间。
- ❑ -w: 当命令执行失败，发出警告，而非默认终止任务。

如果想要通过 Fabric 得知远程机器 192.168.1.205 的 hostname 名，可执行如下命令：

```
fab -p redhat(root密码) -H 192.168.1.205 -- 'hostname'
```

记得在当前目录下用 touch 命令建立一个新的 fabfile.py 文件，不然会产生如下报错：

```
Traceback (most recent call last):
```

```
File "/usr/lib/python2.6/site-packages/Fabric-1.3.0-py2.6.egg/fabric/main.py",
line 600, in main
```

```
arguments, remainder_arguments, default)
```

```
UnboundLocalError: local variable 'default' referenced before assignment
```

成功执行完 fab 命令以后，就可以看得到以下结果了：

```
[192.168.1.205] Executing task '<remainder>'
```

```
[192.168.1.205] run: uname -r
```

```
[192.168.1.205] out: 2.6.32-358.el6.x86_64
```

```
Done.
```

```
Disconnecting from 192.168.1.205... done.
```

3.6.3 Fabric 的核心 API

Fabric 的核心 API 主要有 7 类：带颜色的输出类 (color output)、上下文管理类 (context managers)、装饰器类 (decorators)、网络类 (network)、操作类 (operations)、任务类 (tasks)、

工具类 (utils)。

Fabric 提供了一组操作简单但功能强大的 fabric.api 命令集，简单地调用这些 API 就能完成大部分应用场景的需求，Fabric 支持的常用命令及说明如下。

- ❑ local: 执行本地命令，如 local ('uname -s')。
- ❑ lcd: 切换本地目录，如 lcd ('/home')。
- ❑ cd: 切换远程目录，如 cd ('/data/logs/')。
- ❑ run: 执行远程命令，如 run ('free -m')。
- ❑ sudo: 以 sudo 方式执行远程命令，如 sudo ('/etc/init.d/httpd start')。
- ❑ put: 上传本地文件到远程主机，如 put ('/home/user.info', '/data/user.info')。
- ❑ get: 从远程主机下载文件到本地，如 get ('/home/user.info', '/data/user.info')。
- ❑ prompt: 获得用户输入信息，如 prompt ('please input user password: ')。
- ❑ confirm: 获得提示信息确认，如 confirm ('Test failed, Continue[Y/N]')。
- ❑ reboot: 重启远程主机，如 reboot ()。
- ❑ @task: 函数修饰符。新版本的 Fabric 对面向对象的特性和命名空间有很好的支持。面向对象的继承和多态特性，对代码的复用极其重要。新版本的 Fabric 定义了常规的模块级别的函数，并带有装饰器 @task，这会直接将该函数转化为 task 子类。该函数名会被作为任务名，后面会举例说明 @task 的用法。
- ❑ @runs_once: 函数修饰符。标识此修饰符的函数只会执行一次，不受多台主机影响。

下面来看看 @task 的用法，它可以为任务添加别名，命令如下：

```
from fabric.api import task
@task(alias='dwm')
def deploy_with_migrations():
    pass
```

用 fab 命令打印指定文件中存在的命令，如下：

```
fab -f /home/yhc/test.py --list
```

命令显示结果如下所示：

```
Available commands:
deploy_with_migrations
dwm
```

还可以通过 @task 来设置默认的任务，比如 deploy (部署) 一个子模块，命令如下：

```
from fabric.api import task
@task
def migrate():
    pass
@task
```



```
def push():
    pass
@task
def provision():
    pass
```

```
@task(default=True)
def full_deploy():
    provision()
    push()
    migrate()
```

```
fab -f /home/yhc/test.py --list
```

结果如下所示：

```
Available commands:
  deploy
  deploy.full_deploy
  deploy.migrate
  deploy.provision
  deploy.push
```

也可以通过 `@task` 以类的形式定义任务，例如：

```
from fabric.api import task
from fabric.tasks import Task
class MyTask(Task):
    name = "deploy"
    def run(self, environment, domain="whatever.com"):
        run("git clone foo")
        sudo("service apache2 restart")
instance = MyTask()
```

下面采用 `@task` 方式的代码跟上面的代码效果是一样的：

```
from fabric.api import task
from fabric.tasks import Task
@task
def deploy(environment, domain="whatever.com"):
    run("git clone foo")
    sudo("service apache2 restart")
```

大家可以对比看看，是不是采用 `@task` 函数修饰器的方式更为简洁和直观呢？

关于 `@task` 修饰器的用法和其他 `fabric.api` 命令，请参考 Fabric 官方文档 http://fabric.chs.readthedocs.org/zh_CN/chs/tutorial.html。

这里举个例子说明一下 `@runs_once` 用法，源码文件 `/home/yhc/test.py` 文件内容如下所示：

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from fabric.api import *
```

```

from fabric.colors import *

env.user = "root" #定义用户名, env对象的作用是定义Fabric指定文件的全局设定
env.password = "redhat" #定义密码
env.hosts = ['192.168.1.204', '192.168.1.205']
#定义目标主机

@runs_once
#当有多台主机时只执行一次
def local_task(): #本地任务函数
    local("hostname")
    print red("hello,world")
    #打印红色字体的结果
def remote_task(): #远程任务函数
    with cd("/usr/local/src"):
        run("ls -lF | grep /$")
#with是Python中更优雅的语法, 可以很好地处理上下文环境产生的异常, 这里用了with以后相当于实现了
"cd /var/www/html && ls -lsart"的效果。

```

通过 `fab` 命令调用 `local_task` 本地任务函数, 命令如下:

```
fab -f test.py local_task
```

结果如下所示:

```

[192.168.1.204] Executing task 'local_task'
[localhost] local: hostname
client.cn7788.com
My hostname is client.cn7788.com
Hello,world!
Done.

```

上述命令显示的虽然不是本机的 IP 地址, 但实际上并没有在主机 192.168.1.204 上面执行命令, 而是在本地主机 `client.cn7788.com` (IP 为 192.168.1.206 的机器) 上执行了命令, 并以红色字体显示了“hello,world”和“My hostname is client.cn7788.com”。

调用 `remote_task` 远程函数显示结果, 分别在 204 和 205 的机器上打印 `/usr/local/src/` 下面存在的目录, 结果如下:

```

[192.168.1.204] Executing task 'remote_task'
[192.168.1.204] run: ls -lF | grep /$
[192.168.1.204] out: drwxr-xr-x. 2 root root    4096 Nov 22 00:01 download/
[192.168.1.204] out: drwxr-xr-x. 9  501 games    4096 Nov 19 04:44 Fabric-1.3.0/
[192.168.1.204] out: drwxr-xr-x. 2 root root    4096 Nov 22 00:01 object/
[192.168.1.205] Executing task 'remote_task'
[192.168.1.205] run: ls -lF | grep /$
[192.168.1.205] out: drwxr-xr-x. 2 root root    4096 Nov 22 04:58 mysql/
[192.168.1.205] out: drwxr-xr-x. 2 root root    4096 Nov 22 04:58 puppet/
[192.168.1.205] out: drwxr-xr-x. 2 root root    4096 Nov 22 04:58 soft/
[192.168.1.205] out: drwxr-xr-x. 2 root root    4096 Nov  3 07:56 test/

```

```
Done.
Disconnecting from 192.168.1.204... done.
Disconnecting from 192.168.1.205... done.
```

3.7 Fabric 应用实例

3.7.1 开发环境中的 Fabric 应用实例

笔者公司在开发环境下使用的都是 Xen 和 KVM 虚拟机，有不少数据，因为是内网环境，所以直接用 root 和 SSH 密码连接。系统统一为 CentOS 6.4 x86_64，内核版本为 2.6.32-358.el6.x86_64，Python 版本为 2.6.6。

实例 1，同步 Fabric 跳板机的 /etc/hosts 文件，脚本如下：

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from fabric.api import *
from fabric.colors import *
from fabric.context_managers import *
#fabric.context_managers是Fabric的上下文管理类，这里需要import是因为下面会用到with

env.user = 'root'
env.hosts = ['192.168.1.200', '192.168.1.205', '192.168.1.206']
env.password = 'bilin101'

@task
#限定只有put_hosts_file函数对fab命令可见
def put_hosts_files():
    print yellow("rsync /etc/host File")
    with settings(warn_only=True): #出现异常时继续执行，不终止
        put("/etc/hosts", "/etc/hosts")
        print green("rsync file success!")
    '''这里用到with是确保即便发生异常，也将尽早执行下面的清理操作，一般来说，Python中的with语句一
    般多用于执行清理操作（如关闭文件），因为Python中打开文件以后的时间是不确定的，如果有其他程
    序试图访问打开的文件会导致出现问题。
    '''
    for host in env.hosts:
        env.host_string = host
        put_hosts_files()
```

实例 2，同步公司内部开发服务器的 git 代码，现在互联网公司的开发团队应该都比较倾向于采用 git 作为开发版本管理工具了，此脚本稍微改动下应该也可以应用于线上的机器，脚本如下：

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from fabric.api import *
from fabric.colors import *
```

```

from fabric.context_managers import *

env.user = 'root'
env.hosts = ['192.168.1.200', '192.168.1.205', '192.168.1.206']
env.password = 'redhat'

@task
#同上面一样,指定git_update函数只对fab命令可见
def git_update():
    with settings(warn_only=True):
        with cd('/home/project/github'):
            sudo('git stash clear')
            #清理当前git中所有的储藏,以便于我们stashing最新的工作代码
            sudo('git stash')
            '''如果想切换分支,但是又不想提交正在进行的工作,那么就储藏这些变更。为了往git堆
            栈推送一个新的储藏,只需要运行git stash命令即可
            ...
            sudo('git pull')
            sudo('git stash apply')
            #完成当前代码pull以后,取回最新的stashing工作代码,这里使用命令git stash apply
            sudo('nginx -s reload')

for host in env.hosts:
    env.host_string = host
    git_update()

```

3.7.2 线上环境中的 Fabric 应用实例

笔者线上的核心业务机器统一都是 AWS EC2 主机,机器数量较多,每个数据中心都部署了 Fabric 跳板机(物理拓扑图可参考图 3-3),系统为 Amazon Linux,内核版本为 3.14.34-27.48.amzn1.x86_64,Python 版本为 Python 2.6.9。

如果公司项目组核心开发人员离职,线上机器就都要更改密钥,由于密钥一般是以组的形式存在的,再加上机器数量繁多,因此单纯通过技术人员手工操作,基本上是一项不可能完成的任务,但若是通过 Fabric 自动化运维工具的话,这就是一项简单的工作了,由于现在的线上服务器多采用 SSH Key 的方式管理,所以对于大多数系统运维人员来说 SSH Key 分发也是工作内容之一,故而建议大家掌握此脚本的用法。示例脚本内容如下:

```

#!/usr/bin/python2.6
# -*- coding: utf-8 -*-
from fabric.api import *
from fabric.colors import *
from fabric.context_managers import *
#这里为了简化工作,脚本采用纯Python的写法,没有采用Fabric的@task修饰器

env.user = 'ec2-user'
env.key_filename = '/home/ec2-user/.ssh/id_rsa'
hosts=['budget','adserver','bidder1','bidder2','bidder3','bidder4','bidder5']

```

```

    'bidder6','bidder7','bidder8','bidder9',redis1','redis2','redis3','red
    is4','redis5','redis6']
#机器数量众多, 这里只罗列了部分

def put_ec2_key():
    with settings(warn_only=False):
        put("/home/ec2-user/admin-master.pub", "/home/ec2-user/admin-master.pub")
        sudo("cp /home/ec2-user/admin-master.pub /home/ec2-user/.ssh/authorized_keys")
        #\cp的作用是取消其别名作用, 即不让cp-i生效
        sudo("chmod 600 /home/ec2-user/.ssh/authorized_keys")

def put_admin_key():
    with settings(warn_only=False):
        put("/home/ec2-user/admin-operation.pub",
            "/home/ec2-user/admin-operation.pub")
        sudo("cp /home/ec2-user/admin-operation.pub /home/admin/.ssh/authorized_keys")
        sudo("chown admin:admin /home/admin/.ssh/authorized_keys")
        sudo("chmod 600 /home/admin/.ssh/authorized_keys")

def put_readonly_key():
    with settings(warn_only=False):
        put("/home/ec2-user/admin-readonly.pub",
            "/home/ec2-user/admin-readonly.pub")
        sudo("\cp /home/ec2-user/admin-readonly.pub /home/readonly/.ssh/authorized_keys")
        sudo("chown readonly:readonly /home/readonly/.ssh/authorized_keys")
        sudo("chmod 600 /home/readonly/.ssh/authorized_keys")

for host in hosts:
    env.host_string = host
    put_ec2_key()
    put_admin_key()
    put_readonly_key()

```

大家可以输入如下命令查看系统中定义的别名 (CentOS 6.4 x86_64)。

```
alias
```

命令显示结果如下所示:

```

alias cp='cp -i'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias mv='mv -i'
alias rm='rm -i'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'

```

Amazon Linux 系统与 CentOS 6.4 略有差别, 已经取消了 cp 的别名定义。

如果线上的 Nagios 客户端的监控脚本因为业务需求又发生了改动, 而 bidder 业务集群约有 23 台 (下面只列出了其中 10 台), 且其中的一个业务需求脚本前前后后改动了 4 次, 这时, 手动操作肯定会耗费大量人力及时间成本, 因此这里用 Fabric 推送此脚本并执行,

代码如下：

```
#!/usr/bin/python2.6
## -*- coding: utf-8 -*-
from fabric.api import *
from fabric.colors import *
from fabric.context_managers import *

user = 'ec2-user'
hosts=['bidder1','bidder2','bidder3','bidder4','bidder5','bidder6','bidder7','bi
dder8','bidder9','bidder10']
#机器数量比较多，这里只列出其中10台

@task
#这里用到了@task修饰器
def put_task():
    print yellow("Put Local File to Nagios Client")
    with settings(warn_only=True):
        put("/home/ec2-user/check_cpu_utili.sh",
            "/home/ec2-user/check_cpu_utili.sh")
        sudo("cp /home/ec2-user/check_cpu_utili.sh /usr/local/nagios/libexec")
        sudo("chown nagios:nagios /usr/local/nagios/libexec/check_cpu_utili.sh")
        sudo("chmod +x /usr/local/nagios/libexec/check_cpu_utili.sh")
        sudo("kill `ps aux | grep nrpe | head -n1 | awk '{print $2}'`")
        sudo("/usr/local/nagios/bin/nrpe -c /usr/local/nagios/etc/nrpe.cfg -d")
        print green("upload File success and restart nagios service!")
        #这里以绿色字体打印结果是为了方便查看脚本执行结果

for host in hosts:
    env.host_string = host
    put_task()
```

执行上面的脚本以后，Fabric 也会返回清晰的显示结果，大家可以根据显示结果得知哪些机器已经成功运行，哪些机器失败，非常直观，结果如下所示：

```
Put Local File to remote
[bidder1] put: /home/ec2-user/check_cpu_utili.sh -> /home/ec2-user/check_cpu_utili.sh
[bidder1] sudo: cp /home/ec2-user/check_cpu_utili.sh /usr/local/nagios/libexec/check_cpu_utili.sh
[bidder1] sudo: chown nagios:nagios /usr/local/nagios/libexec/check_cpu_utili.sh
[bidder1] sudo: chmod +x /usr/local/nagios/libexec/check_cpu_utili.sh
[bidder1] sudo: kill `ps aux | grep nrpe | head -n1 | awk '{print $2}'`
[bidder1] sudo: /usr/local/nagios/bin/nrpe -c /usr/local/nagios/etc/nrpe.cfg -d
upload File success and restart nagios service!
Put Local File to remote
[bidder2] put: /home/ec2-user/check_cpu_utili.sh -> /home/ec2-user/check_cpu_utili.sh
[bidder2] sudo: cp /home/ec2-user/check_cpu_utili.sh /usr/local/nagios/libexec/check_cpu_utili.sh
[bidder2] sudo: chown nagios:nagios /usr/local/nagios/libexec/check_cpu_utili.sh
[bidder2] sudo: chmod +x /usr/local/nagios/libexec/check_cpu_utili.sh
[bidder2] sudo: kill `ps aux | grep nrpe | head -n1 | awk '{print $2}'`
[bidder2] sudo: /usr/local/nagios/bin/nrpe -c /usr/local/nagios/etc/nrpe.cfg -d
upload File success and restart nagios service!
```

```

Put Local File to remote
[bidder3] put: /home/ec2-user/check_cpu_utili.sh -> /home/ec2-user/check_cpu_utili.sh
[bidder3] sudo: cp /home/ec2-user/check_cpu_utili.sh /usr/local/nagios/libexec/check_cpu_utili.sh
[bidder3] sudo: chown nagios:nagios /usr/local/nagios/libexec/check_cpu_utili.sh
[bidder3] sudo: chmod +x /usr/local/nagios/libexec/check_cpu_utili.sh
[bidder3] sudo: kill `ps aux | grep nrpe | head -n1 | awk '{print $2}'` `
[bidder3] sudo: /usr/local/nagios/bin/nrpe -c /usr/local/nagios/etc/nrpe.cfg -d
upload File success and restart nagios service!

Put Local File to remote
[bidder4] put: /home/ec2-user/check_cpu_utili.sh -> /home/ec2-user/check_cpu_utili.sh
[bidder4] sudo: cp /home/ec2-user/check_cpu_utili.sh /usr/local/nagios/libexec/check_cpu_utili.sh
[bidder4] sudo: chown nagios:nagios /usr/local/nagios/libexec/check_cpu_utili.sh
[bidder4] sudo: chmod +x /usr/local/nagios/libexec/check_cpu_utili.sh
[bidder4] sudo: kill `ps aux | grep nrpe | head -n1 | awk '{print $2}'` `
[bidder4] sudo: /usr/local/nagios/bin/nrpe -c /usr/local/nagios/etc/nrpe.cfg -d
upload File success and restart nagios service!

Put Local File to remote
[bidder5] put: /home/ec2-user/check_cpu_utili.sh -> /home/ec2-user/check_cpu_utili.sh
[bidder5] sudo: cp /home/ec2-user/check_cpu_utili.sh /usr/local/nagios/libexec/check_cpu_utili.sh
[bidder5] sudo: chown nagios:nagios /usr/local/nagios/libexec/check_cpu_utili.sh
[bidder5] sudo: chmod +x /usr/local/nagios/libexec/check_cpu_utili.sh
[bidder5] sudo: kill `ps aux | grep nrpe | head -n1 | awk '{print $2}'` `
[bidder5] sudo: /usr/local/nagios/bin/nrpe -c /usr/local/nagios/etc/nrpe.cfg -d
upload File success and restart nagios service!

```

大家可以看到，短短几行代码就达到了自动化运维的效果，而且跟 Fabric 相关的代码都是纯 Python 代码和 Shell 代码，开发人员和运维人员很容易上手，在公司里推广应用，大家的认可程度也高。事实上，通过上面的举例大家应该能发现，Fabric 特别适合于需要重复执行大量 Shell 命令的工作场景。

3.8 小结

Fabric 作为 Python 开发的轻量级运维工具，小块头却有大智慧，熟练掌握其用法能够解决工作中的很多自动化运维需求，这应该也是它受到运维人员和开发人青睐的原因。大家可以通过在开发环境和线上环境的应用示例，熟悉掌握相关用法，然后将其应用于自己的系统自动化运维环境。

自动化部署管理工具 Ansible 简介

对比 Puppet 和 Saltstack 而言, Ansible 是一款轻量级的服务器集中管理软件, 它默认采用 SSH 的方式管理客户端, 部署简单, 只需要在跳板机或主控端部署 Ansible 环境, 被控端无需进行任何操作。Ansible 是基于 Python 开发的, 由 Paramiko 和 PyYAML 两个关键模块构建, 它的各种模块可用来实现对客户端进行批量管理(执行命令/安装软件/指定特定任务等), 对于一些较为复杂的需要重复执行的任务, 可以通过 Ansible 下的 playbook 系统来管理。

Ansible 跟第 3 章所讲的 Fabric 一样——都是基于 paramiko 开发的。paramiko 是一个纯 Python 实现的 SSH 协议库。因此 Fabric 和 Ansible 还有一个共同点就是都不需要在远程主机上安装客户端, 因为它们都是基于 SSH 来和远程主机通信的。

关于 Ansible 的更多详细介绍请参考官方网址 <http://www.ansibleworks.com>。

相比较于其他自动化运维工具, Ansible 的优势有很多, 具体如下:

- ❑ 轻量级, 无需在客户端安装 agent, 更新时只需在操作机上进行一次更新即可。
- ❑ 批量任务执行可以写成脚本, 而且不用分发到远程就可以执行。
- ❑ 使用 Python 编写, 维护更简单。
- ❑ 支持 sudo, 能够很好地支持 AWS EC2, 可以很轻松地部署 AWS EC2 环境。
- ❑ Ansible 社区非常活跃。Ansible 本身提供的模块非常丰富, 第三方资源多。

2015 年, 红帽公司 (Red Hat) 宣布收购 Ansible, 在产品层面, Ansible 符合 Red Hat 希望通过开放式开发提供无障碍设计和模块化架构的目标, 主要体现在以下几个方面。

- ❑ Ansible 易于使用: 这点从下面这两个例子可见一斑。一是, Ansible 的 playbook 使用了人类可读的 YAML 代码进行编写, 简化了自动化流程的编写和维护; 二是, Ansible 使用标准的 SSH 连接来执行自动化流程, 不需要代理, 更容易融入已有的企业 IT 环境。
- ❑ Ansible 是模块化的: Ansible 提供了 400 多个模块, 而且还在不断增加, 可用于扩

展该产品的功能。这是 Red Hat 希望在其管理的产品中提供的一个重要功能。

- ❑ Ansible 是一个非常受欢迎的开源项目：在 GitHub 上，Ansible 拥有将近 13 000 颗星和 4000 个分支。另外据 Redmonk 统计，Hacker News 提及 Ansible 的次数也在飞速增长。

在资产组合方面，Ansible 符合 Red Hat 希望的提供多层架构、多层一致性和多供应商支持的目标，主要体现在以下几个方面。

- ❑ Ansible 支持多层部署：按照设计，Ansible 通过 VM 和容器为多层应用程序的部署和配置提供支持。这意味着组织可以将同一应用程序的不同组件自动部署到运行效率最高的层上。比如，Ansible 可以同时为 VMware vSphere 服务器虚拟环境中管理 VM 和客户操作系统，在 OpenStack IaaS 云上部署和管理实例，在 OpenShift PaaS 云上部署应用程序。
- ❑ Ansible 为架构的多个层次带来一致性：借助 Ansible，可以通过编程操作计算架构中从基础设施到应用程序之间的每一层。比如，Ansible 可以自动化包括网络、存储、OS、中间件和应用程序层在内的所有配置工作。
- ❑ Ansible 支持异构 IT 环境：Ansible 可以自动配置来自许多供应商的各种技术，而不只是 Red Hat 的技术。比如，Ansible 既支持 Linux，也支持 Windows；Ansible 使 IT 组织可以管理各种 ISV 和 IHV 技术，比如硬件 F5 BIG-IP 和 Citrix NetScaler 到 Amazon Web 服务和 Google 云计算平台。

参考文档 <http://www.infoq.com/cn/news/2015/10/Red-Hat-DevOps>。

另外，从 Ansible 1.7 版本开始，Ansible 加入了支持管理 Windows 系统的模块，限于篇幅的原因，对此这里不做介绍。有兴趣的朋友可以参考官网 http://docs.ansible.com/ansible/list_of_windows_modules.html。

Ansible 的任务执行流程如图 4-1 所示。

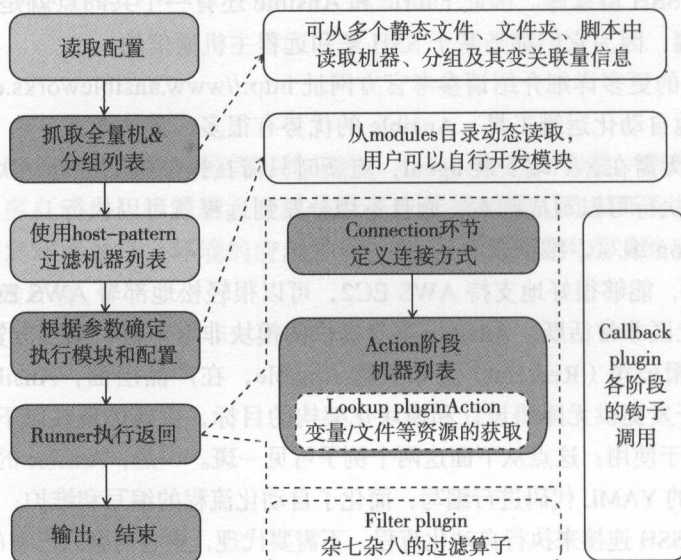


图 4-1 Ansible 任务执行流程图

4.1 YAML 语言介绍

Ansible 里面的配置文件是通过 YAML 文件来实现的，这里首先简单介绍下 YAML 语言。

YAML 是一个可读性高的用来表达资料序列的格式。它的主要特点有：可读性高、语法简单明了、表达能力强、扩展性和通用性强等。

为什么不是我们所熟悉的 XML 呢？因为：

- ❑ YAML 的可读性高。
- ❑ YAML 和脚本语言的交互性好。
- ❑ YAML 使用实现语言的数据类型。
- ❑ YAML 有一个一致的信息模型。
- ❑ YAML 易于实现。

上面 5 条也是 XML 不足的地方。此外，YAML 也有 XML 的下列优点：

- ❑ YAML 可以基于流来处理。
- ❑ YAML 表达能力强，可扩展性好。

总之，YAML 试图用一种比 XML 更敏捷的方式，来完成 XML 所完成的任务。

YAML 的语法和其他高阶语言类似，并且可以简单地表达列表、字典等数据结构。在该语法中，列表中的所有成员都开始于相同的缩进级别，并且使用一个“-”作为开头（一个横杠和一个空格）。一个字典是由一个简单的“键：键值”的形式组成的（这个冒号后面必须是一个空格）。

另外，建议所有的 YAML 文件都是以 ---（3 个横杠）作为开始行，这是 YAML 格式的一部分，表明是一个文件的开始。

下面是一个通用示例，文件内容如下所示：

```
name: Tom Smith
age: 37
spouse:
  name: Jane Smith
  age: 35
children:
  - name: Jimmy Smith
    age: 15
  - name: Jenny Smith
    age: 12
```

上述示例表示 Tom 今年 37 岁，有一个幸福的四口之家，两个孩子 Jimmy 和 Jenny 活泼可爱，妻子 Jane 年轻美貌。

YAML 文件的扩展名通常为 .yaml，如 test.yaml，我们应如何在 Python 下读取 test.yaml 文件呢？代码如下所示：


```
#!/usr/bin/python
#加载YAML模块
import yaml
#读取test.yaml文件
file = open("test.yaml")
#导入文件
x = yaml.load(file)
print x
```

执行这段代码，结果如下所示：

```
{'age': 37, 'spouse': {'age': 25, 'name': 'Jane Smith'}, 'name': 'Tom Smith',
  'children': [{'age': 15, 'name': 'Jimmy Smith'}, {'age': 12, 'name': 'Jenny
  Smith'}]}
```

YAML 文件最常见的层次和结构，对应的就是 Python 中的列表 (list) 和字典 (dictionary) 两种类型，下面举例说明：

```
- apple
- banana
- orange
- pear
```

对应的 Python 结果为：

```
['apple', 'banana', 'orange', 'pear']
```

稍为复杂的 YAML 示例，文件内容如下：

```
list
node_a:
  conntimeout: 300
  external:
  iface: eth0
  port: 556
  internal:
  iface: eth0
  port: 778
  broadcast:
  client: 1000
  server: 2000
node_b:
  0:
  ip: 10.0.0.1
  name: b1
  1:
  ip: 10.0.0.2
  name: b2
```

对应的 Python 结果为：

```
{'node_b': {0: None, 'ip': '10.0.0.2', 'name': 'b2', 1: None}, 'node_a':
  {'iface': 'eth0', 'port': 778, 'server': 2000, 'broadcast': None, 'client':
  1000, 'external': None, 'conntimeout': 300, 'internal': None}}
```

列表和字典结构也可以混用，例如下面的例子：

```
---
# 一位职工记录
name: Example Developer
job: Developer
skill: Elite
employed: True
foods:
  - Apple
  - Orange
  - Strawberry
  - Mango
languages:
  ruby: Elite
  python: Elite
  dotnet: Lame
```

对应的 Python 结果为：

```
{'languages': {'python': 'Elite', 'dotnet': 'Lame', 'ruby': 'Elite'}, 'foods':
  ['Apple', 'Orange', 'Strawberry', 'Mango'], 'name': 'Example Developer',
  'employed': True, 'skill': 'Elite', 'job': 'Developer'}
```

参考文档 <https://www.ibm.com/developerworks/cn/xml/x-1103linrr/>

参考文档 <http://docs.ansible.com/ansible/YAMLSyntax.html>

4.2 Ansible 的安装步骤

Ansible 目前的应用比较广泛，内网开发环境和 AWS 云平台上面都有其应用。这里将以内网开发环境来说明下，系统版本均为 CentOS 6.4 x86_64，Python 版本为 2.6.6。

内网环境机器分配情况如下。

192.168.1.207 主机名：Ansiable.example.com，作用：Ansible 主控端。

192.168.1.205 主机名：client1.example.com，作用：Ansible 被控端机器。

192.168.1.206 主机名：client2.example.com，作用：Ansible 被控端机器。

Ansible 的安装非常简单，安装步骤如下。

1) 下载 epel 源，并通过其安装 Ansible，命令如下：

```
cd /usr/local/src
wget http://ftp.linux.ncsu.edu/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
rpm -ivh epel-release-6-8.noarch.rpm
```

2) 只需要在主控端机器上安装即可，其他被控端机器无需任何操作，命令如下：

```
yum -y install ansible
```

3) 安装成功以后，可以通过命令查看 Ansible 的当前版本，命令如下：

```
ansible --version
```

命令显示结果如下所示：

```
ansible 1.9.4
  configured module search path = None
```

这里显示 Ansible 的当前版本为 1.9.4。

这是一切都顺利的情况，另外一台机器，由于将 Python 版本升级至 2.7.9，因此在运行 Ansible 时报错如下：

```
Traceback (most recent call last):
  File "/usr/bin/ansible", line 36, in <module>
    from ansible.runner import Runner
ImportError: No module named ansible.runner
```

这是因为 Ansible 还是基于原先 Python2.6.6 版本安装的，所以执行 Ansible 时会报错，此时将 /usr/bin/ansible 文件中的第一行解释器 #!/usr/bin/python 改回原版 #!/usr/bin/python2.6 即可。

4) 将两台 client 机器添加进 Ansible 的 webserver 组。

首先我们注释掉 /etc/ansible/hosts 文件里的所有内容，直接在 vim 里执行如下操作即可：

```
%s@^@#@
```

然后添加下列内容：

```
192.168.1.205
192.168.1.206
[webserver]
192.168.1.205
192.168.1.206
```

5) 定义主机与组规则 (Inventory)

Ansible 通过定义好的主机与组规则 Inventory 指定了 Ansible 起作用的主机列表，Ansible 默认读取 /etc/ansible/hosts 文件。下面是 Inventory 文件的一个例子：

```
mail.example.com
[webservers]
foo.example.com
bar.example.com
[dbservers]
one.example.com
two.example.com
Three.example.com
```

如果是通过 pip 安装的 Ansible，则没有默认的 /etc/ansible/hosts 文件，需要手动建立，笔者一般是建立在自己的工作目录下面，比如 /home/yhc/ansible/hosts，然后执行 ansible 命

令时通过 `-i` 参数指定安装目录，如下所示：

```
ansible -i /home/yhc/ansible/hosts bidder -m ping
```

其中，中括号内是组名称，一台主机可以属于多个组。一台属于多个组的主机会读取多个组的变量文件，这样可能会产生冲突，作为解决方案的优先级会在后面介绍。

有一个主机会被 Ansible 默认地自动添加到 Inventory 中，那就是 localhost。Ansible 认为 localhost 就代表本地主机，所以需要它的时候会直接在本机执行而不通过 SSH 连接。

如果 SSH 使用的不是 SSH 默认端口，可以在主机后面指定 SSH 端口，命令如下所示：

```
badwolf.example.com:5309
```

如果使用静态 IP，希望在 hosts 文件中使用别名或通过通道连接，可以采用类似如下的方式：

```
jumper ansible_ssh_port=5555 ansible_ssh_host=192.168.1.50
```

如果有很多主机名称类似，则没有必要一一列出，例如：

```
[webservers]
www[01:50].example.com
db-[a:f].example.com
```

其中数字开头的 0 可以省略，中括号是闭合的。

也可以指定每个主机的连接类型和用户名：

```
[targets]
localhost ansible_connection=local
other1.example.com ansible_connection=ssh ansible_ssh_user=mpdehaan
other2.example.com ansible_connection=ssh ansible_ssh_user=mdhehaan
```

上面这些直接在 Inventory 文件中添加参数的方式并不是一个很好的选择，后面会介绍更好的方法，就是在单独的 `host_vars` 目录中定义参数。

另外，通过主机变量和组变量的方式可以让 Inventory 文件更加灵活。

□ 主机变量

主机可以指定变量，以便后面供 playbook 配置使用，例如下面定义了主机 `host1` 和 `host2` 上 Apache 的参数 `http_port` 及 `maxRequestsPerChild`。

```
[atlanta]
host1 http_port=80 maxRequestsPerChild=808
host2 http_port=303 maxRequestsPerChild=909
```

□ 组变量

组变量的作用是覆盖组中的所有成员，通过定义一个新块，块名由组名 + “`:vars`” 组成，如下例所示：

```
[atlanta]
host1
```

```
host2
[atlanta:vars]
ntp_server=ntp.atlanta.example.com
proxy=proxy.atlanta.example.com
```

❑ 组的组（也称为组嵌套）

组嵌套是通过定义一个新块，块名由组名 + “:children” 组成，如下例所示：

```
[atlanta]
host1
host2
[raleigh]
host2
host3
[southeast:children]
atlanta
raleigh
[usa:children]
southeast
northeast
southwest
Northwest
```

❑ 分离主机和组变量

为了更好地规范定义的主机和组变量，Ansible 支持将 hosts 文件定义的主机名与组变量单独分离出来，并采用 YAML 格式存放到指定的文件中。

假设 Inventory 文件的路径为 /etc/ansible/hosts，其中有个主机名为 foosbal，属于 raleigh 和 webservers 两个组，那么以下位置的 YAML 文件会对 foosball 主机有效：

```
/etc/ansible/group_vars/raleigh
/etc/ansible/group_vars/webservers
/etc/ansible/host_vars/foosball
```

例如，/etc/ansible/group_vars/raleigh 文件可能看起来类似于下面这样：

```
---
ntp_server: acme.example.org
database_server: storage.example.org
```

Ansible 1.2 及之后的版本中，group_vars 和 host_vars 目录既可以在 playbook 目录下也可以在 Inventory 目录下，如果两者都有，playbook 目录下的会覆盖 Inventory 目录下的^①。

6) 测试 Ansible 安装是否成功。

分别执行下面的两条命令：

```
ansible 192.168.1.205 -m ping -k
SSH password:
ansible 192.168.1.206 -m ping -k
```

① 参考文档：<http://docs.ansible.com/ansible/index.html>。

SSH password:

由于 Ansible 主控端和被控端暂时未配置 SSH 证书信任关系，所以需要在执行 Ansible 命令时输入 -k 参数，此时需要提供客户端的 root 密码，最后结果显示如下：

```
192.168.1.205 | success >> {
  "changed": false,
  "ping": "pong"
}
192.168.1.206 | success >> {
  "changed": false,
  "ping": "pong"
}
```

如果出现以上结果，则表示 Ansible 已经成功安装，并且跟客户端机器的连接也是成功的。

4.3 利用 ssh-keygen 设置 SSH 无密码登录

线上的 AWS 云计算平台基于自动化运维的原则，Ansible 也被部署在跳板机上，关于跳板机的介绍请大家参考第3章的内容，这里不再重复。其物理拓扑图如图4-2所示。



图 4-2 跳板机物理拓扑图

为了方便自动化运维，在 Ansible 跳板机上用 ssh-keygen 设置 SSH 无密码登录其他客户端机器是很有必要的，具体操作步骤如下。

1) 首先用命令生成一对密钥，命令如下：

```
ssh-keygen -t rsa
```

命令显示结果如下所示：

```
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa): (回车)
Created directory '/root/.ssh'.
Enter passphrase (empty for no passphrase): (回车)
Enter same passphrase again: (回车)
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
```

The key fingerprint is:

60:7b:a4:80:de:0d:55:d7:14:ee:39:fa:fd:c0:4a:cc root@Ansible.example.com

The key's randomart image is:

```

+--[ RSA 2048 ]-----+
|      ... ..oo.      |
|      . . . . .      |
|      . o o . .      |
|      . . = = . .      |
|      . . + S +      |
|      .      + o      |
|      .      E o      |
|      .      o o .      |
|      .      o ...      |
+-----+

```

2) 然后用 `ssh-copy-id` 命令将公钥分别下发到 `client1` 和 `client2` 机器上, 命令如下:

```
ssh-copy-id -i /root/.ssh/id_rsa.pub root@192.168.1.205
```

`client1` 机器结果如下:

```

The authenticity of host '192.168.1.205 (192.168.1.205)' can't be established.
RSA key fingerprint is 8d:72:e5:fa:5a:c6:c1:e2:e1:00:bc:8d:6a:6f:2b:3a.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.205' (RSA) to the list of known hosts.
root@192.168.1.205's password:
Now try logging into the machine, with "ssh 'root@192.168.1.205'", and check in:
.ssh/authorized_keys
to make sure we haven't added extra keys that you weren't expecting.

```

需要说明的是, 第一次运行时, 要先输入一下 “yes” 进行公钥验证, 后续无需再次输入。

```
ssh-copy-id -i /root/.ssh/id_rsa.pub root@192.168.1.206
```

`client2` 机器结果如下:

```

The authenticity of host '192.168.1.206 (192.168.1.206)' can't be established.
RSA key fingerprint is 8d:72:e5:fa:5a:c6:c1:e2:e1:00:bc:8d:6a:6f:2b:3a.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.206' (RSA) to the list of known hosts.
root@192.168.1.206's password:
Now try logging into the machine, with "ssh 'root@192.168.1.206'", and check in:
.ssh/authorized_keys

```

上面的步骤执行完毕以后, 可以分别执行下面的命令进行验证:

```
ssh 192.168.1.205
ssh 192.168.1.206
```

因为这里本身就是以 `root` 账户执行操作的, 所以无需以 `root@192.168.1.205` 的命令来执行, 如果能直接以无密码进入目标主机就说明公钥分发成功, 整个配置过程是没有问题的。

如果是 AWS EC2 机器，那么默认是不允许 root 连接的（只允许具有 sudo 权限的 ec2-user 用户），因此操作起来稍微麻烦一些（copy 模块的用法下面会有介绍）。先查看当前用户，命令如下：

```
$ whoami
ec2-user
```

然后以 ec2-user 用户的身份执行 Ansible 命令，如下：

```
$ansible bidder -m copy -a "src=/usr/local/src/nagios-server.sh dest=/tmp/
owner=root group=root mode=0644 force=yes" --sudo
```

这里稍微说明下，因为之前已经调试好了 nagios-server.sh，是利用 root 用户来进行调试的，所以这里需要加上 --sudo，ec2-user 用户是具有 sudo 权限的。



注意 最后的 --sudo 并非是 -sudo，此处要么用 -s，要么用 --sudo，不然命令是会报错的。

4.4 Ansible 常用模块介绍

Ansible 有很多模块，包括云计算、命令行、包管理、系统服务、用户管理等，可以通过官方网站 http://docs.ansible.com/modules_by_category.html 查看相应的模块，也可以在命令行下通过 `ansible-doc -l` 命令查看模块，或者通过 `ansible-doc -s` 模块名查看具体某个模块的使用方法。官网的介绍比较详细，建议查看官网介绍。`ansible-doc -l` 命令的部分显示结果如下所示：

```
less comes with NO WARRANTY, to the extent permitted by law.
For information about the terms of redistribution,
see the file named README in the less distribution.
Homepage: http://www.greenwoodsoftware.com/less

a10_server          Manage A10 Networks AX/SoftAX/Thunder/vThunder devices
a10_service_group   Manage A10 Networks AX/SoftAX/Thunder/vThunder devices
a10_virtual_server  Manage A10 Networks AX/SoftAX/Thunder/vThunder devices
acl                 Sets and retrieves file ACL information.
add_host            add a host (and alternatively a group) to the ansible-
                    playbook in-memory inventory
airbrake_deployment Notify airbrake about app deployments
alternatives        Manages alternative programs for common commands
apache2_module      enables/disables a module of the Apache2 webserver
apt                 Manages apt-packag
apt_key             Add or remove an apt key
apt_repository      Add and remove APT repositories
apt_rpm             apt_rpm package manager
assemble            Assembles a configuration file from fragments
assert             Fail with custom message
at                 Schedule the execution of a command or script file via
                    the at command.
```

authorized_key	Adds or removes an SSH authorized key
azure	create or terminate a virtual machine in azure
bigip_facts	Collect facts from F5 BIG-IP devices
bigip_monitor_http	Manages F5 BIG-IP LTM http monitors
bigip_monitor_tcp	Manages F5 BIG-IP LTM tcp monitors
bigip_node	Manages F5 BIG-IP LTM nodes
bigip_pool	Manages F5 BIG-IP LTM pools
bigip_pool_member	Manages F5 BIG-IP LTM pool members
bigpanda	Notify BigPanda about deployments
boundary_meter	Manage boundary meters
bower	Manage bower packages with bower
bzr	Deploy software (or files) from bzr branches
campfire	Send a message to Campfire
capabilities	Manage Linux capabilities



注意 Ansible 的模块实现的行为是幂等性 (idempotence) 的, 只需要运行一次 playbook 就可以将需要配置的机器都置为期望状态。这是一个非常赞的特性, 因为它意味着向同一台机器多次执行一个 playbook 是安全的。

Ansible 命令行调用模块的语法如下:

ansible 操作目标 -m 模块名 -a 模块参数

下面将分别介绍运维工作中经常会用到 13 个模块, 其他模块在本书中就不再一一介绍了, 具体详情建议大家参考官方文档。

- ☐ setup 模块
- ☐ copy 模块
- ☐ synchronize 模块
- ☐ file 模块
- ☐ ping 模块
- ☐ group 模块
- ☐ user 模块
- ☐ shell 模块
- ☐ script 模块
- ☐ get_url 模块
- ☐ yum 模块
- ☐ cron 模块
- ☐ service 模块

1. setup 模块

该模块可用于获取 Ansible 客户端机器的详细信息, 命令如下:

```
ansible webserver(机器组名) -m setup
```

命令显示的部分结果如下(完整结果太详细了,这里只截取了部分显示):

```
192.168.1.206 | success >> {
  "ansible_facts": {
    "ansible_all_ipv4_addresses": [
      "192.168.1.206"
    ],
    "ansible_all_ipv6_addresses": [
      "fe80::216:3eff:fe08:ea2b"
    ],
    "ansible_architecture": "x86_64",
    "ansible_bios_date": "",
    "ansible_bios_version": "",
    "ansible_cmdline": {
      "KEYTABLE": "us",
      "LANG": "en_US.UTF-8",
      "SYSFONT": "latarcyrheb-sun16",
      "console": "hvc0",
      "quiet": true,
      "rd_LVM_LV": "VolGroup/lv_root",
      "rd_NO_DM": true,
      "rd_NO_LUKS": true,
      "rd_NO_MD": true,
      "rhgb": true,
      "ro": true,
      "root": "/dev/mapper/VolGroup-lv_root"
    },
    "ansible_date_time": {
      "date": "2015-11-25",
      "day": "25",
      "epoch": "1448418484",
      "hour": "02",
      "iso8601": "2015-11-25T02:28:04Z",
      "iso8601_micro": "2015-11-25T02:28:04.467675Z",
      "minute": "28",
      "month": "11",
      "second": "04",
      "time": "02:28:04",
      "tz": "UTC",
      "tz_offset": "+0000",
      "weekday": "Wednesday",
      "year": "2015"
    }
  }
}
```

2. copy 模块

该模块可实现 Ansible 主机向客户端传送文件的功能,类似于 scp,请大家记得提前关闭所有机器的 SELinux 功能,不然会出现如下报错:


```

192.168.1.205 | FAILED >> {
  "checksum": "d3869c634275c17b9a0561b1f9ac02f685353a53",
  "failed": true,
  "msg": "Aborting, target uses selinux but python bindings (libselinux-python)
        aren't installed!"
}

192.168.1.206 | FAILED >> {
  "checksum": "d3869c634275c17b9a0561b1f9ac02f685353a53",
  "failed": true,
  "msg": "Aborting, target uses selinux but python bindings (libselinux-python)
        aren't installed!"
}

```

如果出现上述错误，我们该如何修复呢？需要在被控端安装 `libselinux-python` 包来进行修复，命令如下所示：

```
ansible webserver -m command -a "yum -y install libselinux-python"
```

修复错误以后，再次输入 `copy` 模块的命令，如下：

```
ansible webserver -m copy -a "src=/usr/local/src/test.py dest=/tmp/ owner=root
group=root mode=0755 force=yes"
```

其他参数都比较好理解，这里解释下 `force` 参数和 `backup` 参数。

❑ **force**：如果目标主机包含该文件，但内容不同，则设置为 `yes` 后会强制覆盖，设置为 `no` 后，只有当目标主机的目标位置不存在该文件时，才复制该文件到目标主机；默认值为 `yes`。

❑ **backup**：在覆盖之前备份源文件，备份文件包含时间。该参数有两个选项 `yes` 和 `no`。命令显示结果如下所示：

```

192.168.1.206 | success >> {
  "changed": false,
  "checksum": "da39a3ee5e6b4b0d3255bfef95601890afd80709",
  "dest": "/tmp/test.py",
  "gid": 0,
  "group": "root",
  "mode": "0755",
  "owner": "root",
  "path": "/tmp/test.py",
  "size": 0,
  "state": "file",
  "uid": 0
}

```

```


192.168.1.205 | success >> {
  "changed": false,
  "checksum": "da39a3ee5e6b4b0d3255bfef95601890afd80709",
  "dest": "/tmp/test.py",

```

```

"gid": 0,
"group": "root",
"mode": "0755",
"owner": "root",
"path": "/tmp/test.py",
"size": 0,
"state": "file",
"uid": 0
}

```

 **注意** copy 模块跟 rsync 命令一样，如果路径使用 “/” 来结尾，则只复制目录里的内容；如果没有使用 “/” 来结尾，则包含目录在内的整个内容全部被复制。

3. synchronize 模块

由于 synchronize 模块会调用 rsync 命令，因此首先要记得提前安装好 rsync 软件包。synchronize 模块用于将 Ansible 机器的指定目录推送（push）到客户机器的指定目录下，命令如下：

```

ansible 192.168.1.206 -m synchronize -a "src=/usr/local/src/ dest=/usr/local/src/
delete=yes compress=yes "

```

显示结果如下所示：

```

192.168.1.206 | success >> {
  "changed": true,
  "cmd": "rsync --delay-updates -F --compress --delete-after --archive --rsh
'ssh -S none -o StrictHostKeyChecking=no' --out-format='<<CHANGED>>%i
%n%L' \"/usr/local/src/" \"root@192.168.1.206:/usr/local/src/\"",
  "msg": ".d..t..... ./\n<f+++++++ epel-release-6-8.noarch.rpm\n<f+++++++
limit.sh\n<f+++++++ test.py\n",
  "rc": 0,
  "stdout_lines": [
    ".d..t..... ./",
    "<f+++++++ epel-release-6-8.noarch.rpm",
    "<f+++++++ limit.sh",
    "<f+++++++ test.py"
  ]
}

```

其中，delete=yes 用来实现使两边的内容一样（即以 push 方式为主），实现效果跟 rsync --delete 效果一样，如果是客户端不存在的文件或目录则增补，如果存在着不同的文件或目录则删除，以保证两边内容一致。

compress=yes 用于开启压缩，默认为开启。

另外，由于 synchronize 模块调用的是 rsync 命令，因此如果路径使用 “/” 来结尾，则只复制目录里的内容；如果没有使用 “/” 来结尾，则包含目录在内的整个内容全部被复制。

4. file 模块

file 模块主要用来设置文件或目录的属性。

- ❑ group: 定义文件或目录的属组。
- ❑ mode: 定义文件或目录的权限。
- ❑ owner: 定义文件或目录的属主。
- ❑ path: 必选项, 定义文件或目录的路径。
- ❑ recurse: 递归设置文件的属性, 只对目录有效。
- ❑ src: 被链接的源文件路径, 只应用于 state=link 的情况。
- ❑ dest: 被链接到的路径, 只应用于 state=link 的情况。
- ❑ force: 强制创建软链接, 有两种情况, 一种是源文件不存在, 但之后会建立的情况; 另一种是要先取消已创建的软链接, 再创建新的软链接, 有两个选项 yes 和 no。
- ❑ state: 后面连接文件的各种状态, 例如下面的 directory、link、hard、file 及 absent。
- ❑ link: 创建软链接。
- ❑ hard: 创建硬链接。
- ❑ directory: 如果目录不存在, 则创建目录。
- ❑ file: 即使文件不存在, 也不会被创建。
- ❑ absent: 删除目录、文件或链接文件。
- ❑ touch: 如果文件不存在, 则会创建一个新的文件, 如果文件或目录已存在, 则更新其最后的修改时间, 这一点跟 Linux 的 touch 命令效果是一样的。

示例一: 将客户端机器 192.168.1.205 的 /usr/local/src/test.py 软链接到 /tmp/test.py 下, 命令如下:

```
ansible 192.168.1.205 -m file -a "src=/usr/local/src/test.py dest=/tmp/test.py
state=link"
```

命令显示结果如下:

```
192.168.1.205 | success >> {
  "changed": true,
  "dest": "/tmp/test.py",
  "gid": 0,
  "group": "root",
  "mode": "0777",
  "owner": "root",
  "size": 22,
  "src": "/usr/local/src/test.py",
  "state": "link",
  "uid": 0
}
```

若要直接在 Ansible 机器上查看 205 机器是否存在 /tmp/test.py 文件, 可采用如下命令:

```
ansible 192.168.1.205 -m command -a 'll /tmp/test.py'
```

命令显示结果如下:

```
192.168.1.205 | success | rc=0 >>
lrwxrwxrwx 1 root root 22 Nov 25 09:13 /tmp/test.py -> /usr/local/src/test.py
```

示例二: 将刚刚建立的 /tmp/test.py 链接文件删除, 命令如下:

```
absible 192.168.1.205 -m file -a "path=/tmp/test.py state=absent"
```

命令显示结果如下:

```
192.168.1.205 | success >> {
  "changed": true,
  "path": "/tmp/test.py",
  "state": "absent"
```

现在再查看下是否还存在着 /tmp/test.py 文件, 命令如下:

```
ansible 192.168.1.205 -m command -a 'll /tmp/test.py'
```

结果为:

```
192.168.1.205 | FAILED | rc=2 >>
ls: cannot access /tmp/test.py: No such file or directory
```

结果报错, 显示不存在此文件, 说明已经成功将其删除了。

示例三: 在 webserver 组建立 /text.txt 文件, 属主和属组均为 root, 权限为 0755, 命令如下:

```
ansible webserver -m file -a 'path=/text.txt state=touch owner=root group=root
mode=0755'
```

命令显示结果如下所示:

```
192.168.1.205 | success >> {
  "changed": true,
  "dest": "/text.txt",
  "gid": 0,
  "group": "root",
  "mode": "0755",
  "owner": "root",
  "size": 0,
  "state": "file",
  "uid": 0
```

```
192.168.1.206 | success >> {
  "changed": true,
  "dest": "/text.txt",
  "gid": 0,
  "group": "root",
  "mode": "0755",
```

```

    "owner": "root",
    "size": 0,
    "state": "file",
    "uid": 0
}

```

示例四：在 **webserver** 组建立 **test** 目录，属主和属组均为 **root**，权限为 **0755**，命令如下：

```

ansible webserver -m file -a 'path=/tmp/test state=directory owner=root group=root
mode=0755'

```

命令显示结果如下所示：

```

192.168.1.205 | success >> {
  "changed": true,
  "gid": 0,
  "group": "root",
  "mode": "0755",
  "owner": "root",
  "path": "/tmp/test",
  "size": 4096,
  "state": "directory",
  "uid": 0
}

```

```

192.168.1.206 | success >> {
  "changed": true,
  "gid": 0,
  "group": "root",
  "mode": "0755",
  "owner": "root",
  "path": "/tmp/test",
  "size": 4096,
  "state": "directory",
  "uid": 0
}

```

5. ping 模块

这个模块前文曾多次提及，用于检测与被控端机器的连通性，命令如下：

```

ansible webserver -m ping

```

6. group 模块

group 模块可以在所有节点上创建自己定义的组，比如利用此模块创建一个组名为 **test**、**gid** 为 **2016** 的组，命令如下：

```

ansible webserver -m group -a 'gid=2016 name=test'

```

命令显示结果如下所示：


```
192.168.1.206 | success >> {
  "changed": true,
  "gid": 2016,
  "name": "test",
  "state": "present",
  "system": false
}
```

```
192.168.1.205 | success >> {
  "changed": true,
  "gid": 2016,
  "name": "test",
  "state": "present",
  "system": false
}
```

现在可以查看下是否已经正常创建这个名为 test 的组了，执行如下命令：

```
ansible webserver -m shell -a 'cat /etc/group | grep test'
```

命令显示结果如下所示：

```
192.168.1.206 | success | rc=0 >>
test:x:2016:
192.168.1.205 | success | rc=0 >>
test:x:2016:
```

现在，我们可以看到两台机器都有组名为 test、gid 为 2016 的组了。



注意 此处用到了 Shell 模块，而没有使用默认的 command 模块，是因为 Shell 模块支持管道符命令，如果此处还是沿用 command 模块的话是要报错的。

7. user 模块

该模块用于创建用户。在指定的节点上创建一个用户名为 test、组为 test 的用户，命令如下：

```
ansible webserver -m user -a "name=test group=test"
```

命令显示结果如下：

```
192.168.1.205 | success >> {
  "changed": true,
  "comment": "",
  "createhome": true,
  "group": 100,
  "groups": "test",
  "home": "/home/test",
  "name": "test",
  "shell": "/bin/bash",
}
```

```

    "state": "present",
    "system": false,
    "uid": 501
}

```

```

192.168.1.206 | success >> {
  "changed": true,
  "comment": "",
  "createhome": true,
  "group": 100,
  "groups": "test",
  "home": "/home/test",
  "name": "test",
  "shell": "/bin/bash",
  "state": "present",
  "system": false,
  "uid": 503
}

```

删除此用户 test，可用如下命令：

```
ansible webserver -m user -a "name=test state=absent remove=yes"
```

命令显示结果如下所示：

```

192.168.1.205 | success >> {
  "changed": true,
  "force": false,
  "name": "test",
  "remove": true,
  "state": "absent"
}

```

```

192.168.1.206 | success >> {
  "changed": true,
  "force": false,
  "name": "test",
  "remove": true,
  "state": "absent"
}

```

8. shell 模块

command 模块作为 Ansible 的默认模块，可以运行被控端机器权限范围内的所有 Shell 命令，前面已多次提到，这里不再重复。而 Shell 模块是执行被控端机器的 Shell 脚本文件，跟另一个模块 raw 的功能类似，并且支持管道符。

比如要执行 webserver 组机器下的 /tmp/echo_hello.sh 文件，命令如下：

```
ansible webserver -m shell -a "/tmp/echo_hello.sh"
```

显示结果如下：

```
192.168.1.205 | success | rc=0 >>
hello,world
```

```
192.168.1.206 | success | rc=0 >>
hello,world
```

9. script 模块

script 模块用于在远程被控端主机上执行本地 Ansible 机器中的 Shell 脚本文件，相当于 scp+shell 的组合命令，比如，要执行本地机器的 /root/print_hello.sh，命令如下：

```
ansible webserver -m script -a "/root/print_hello.sh"
```

命令显示结果如下所示：

```
192.168.1.205 | success >> {
  "changed": true,
  "rc": 0,
  "stderr": "OpenSSH_5.3p1, OpenSSL 1.0.1e-fips 11 Feb 2013\ndebug1: Reading
configuration data /etc/ssh/ssh_config\ndebug1: Applying options for *\n
r\ndebug1: auto-mux: Trying existing master\nr\ndebug1: mux_client_request_
session: master session id: 2\nr\ndebug1: mux_client_request_session: master
session id: 2\nr\nShared connection to 192.168.1.205 closed.\n",
  "stdout": "hello,world\n"}
}
```

```
192.168.1.206 | success >> {
  "changed": true,
  "rc": 0,
  "stderr": "OpenSSH_5.3p1, OpenSSL 1.0.1e-fips 11 Feb 2013\ndebug1: Reading
configuration data /etc/ssh/ssh_config\ndebug1: Applying options for *\n
r\ndebug1: auto-mux: Trying existing master\nr\ndebug1: mux_client_request_
session: master session id: 2\nr\ndebug1: mux_client_request_session: master
session id: 2\nr\nShared connection to 192.168.1.206 closed.\n",
  "stdout": "hello,world\n"}
}
```

10. get_url 模块

get_url 模块可以实现在远程主机上下载 url 到本地，这个模块应该在平时的工作中用得比较多，比如，webserver 组的被控端机器需要下载 http://ftp.linux.ncsu.edu/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm 文件到 /tmp 目录下，命令如下：

```
ansible webserver -m get_url -a 'url= http://ftp.linux.ncsu.edu/pub/epel/6/x86_64/
epel-release-6-8.noarch.rpm dest=/tmp'
```

命令显示结果如下所示：

```
192.168.1.206 | success >> {
  "changed": true,
  "checksum": "2b2767a5ae0de30b9c7b840f2e34f5dd9deaf19a",
  "dest": "/tmp/epel-release-6-8.noarch.rpm",
```

```

"gid": 0,
"group": "root",
"md5sum": "2cd0ae668a585a14e07c2ea4f264d79b",
"mode": "0644",
"msg": "OK (14540 bytes)",
"owner": "root",
"sha256sum": "",
"size": 14540,
"src": "/tmp/tmp41ZkbI",
"state": "file",
"uid": 0,
"url": "http://ftp.linux.ncsu.edu/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm"
}

```

```

192.168.1.205 | success >> {
  "changed": true,
  "checksum": "2b2767a5ae0de30b9c7b840f2e34f5dd9deaf19a",
  "dest": "/tmp/epel-release-6-8.noarch.rpm",
  "gid": 0,
  "group": "root",
  "md5sum": "2cd0ae668a585a14e07c2ea4f264d79b",
  "mode": "0644",
  "msg": "OK (14540 bytes)",
  "owner": "root",
  "sha256sum": "",
  "size": 14540,
  "src": "/tmp/tmp5j3tu5",
  "state": "file",
  "uid": 0,
  "url": "http://ftp.linux.ncsu.edu/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm"
}

```

11. yum 模块

顾名思义，yum 模块是用来管理 Linux 平台的软件包操作的。

- ❑ **config_file**: yum 的配置文件。
- ❑ **disable_gpg_check**: 关闭 gpg_check。
- ❑ **disablerepo**: 不启用某个源。
- ❑ **enablerepo**: 启用某个源。
- ❑ **name**: 要进行操作的软件包的名字，也可以传递一个 url 或一个本地的 rpm 包的路径。
- ❑ **state**: present|absent|latest，state 状态对应了 3 种软件包状态，这里 present 和 latest 都表示安装，而 absent 则表示移除。

若要在指定的被控端机器 192.168.1.206 上面用 Nginx 的 yum 源安装 Nginx 软件包，可采用如下命令：

```
ansible 192.168.1.206 -m yum -a 'name=nginx enablerepo=nginx state=present'
```

显示结果如下:

```
192.168.1.206 | success >> {
  "changed": true,
  "msg": "",
  "rc": 0,
  "results": [
    "Loaded plugins: fastestmirror\nSetting up Install Process\nLoading mirror
    speeds from cached hostfile\n * base: mirrors.163.com\n * extras: mirror.
    bit.edu.cn\n * updates: centos.ustc.edu.cn\nResolving Dependencies\n-->
    Running transaction check\n--> Package nginx.x86_64 0:1.8.0-1.el6.ngx
    will be installed\n--> Finished Dependency Resolution\n\nDependencies
    Resolved\n\n=====
    \n Package                Arch                Version
    Repository              =====
    \nInstalling:\n nginx                x86_64                1.8.0-1.el6.
    ngx                    nginx                352 k\n\nTransaction Summary\n=====
    \nInstall                1
    Package(s)\n\nTotal download size: 352 k\nInstalled size: 872 k\nDownloading
    Packages:\nRunning rpm_check_debug\nRunning Transaction Test\nTransaction Test
    Succeeded\nRunning Transaction\n\nr Installing : nginx-1.8.0-1.el6.ngx.x86_64
    1/1 \n-----
    \n\nThanks for using nginx!\n\nPlease find the official documentation for nginx
    here:\n* http://nginx.org/en/docs/\n\nCommercial subscriptions for nginx are
    available on:\n* http://nginx.com/products/\n\n-----
    \n\nr Verifying : nginx-1.8.0-1.el6.ngx.x86_6
    4
    1/1 \n\nInstalled:\n nginx.x86_64 0:1.8.0-
    1.el6.ngx
    \n\nComplete!\n"
  ]
}
```

可以通过如下命令查看软件包是否通过 Nginx 源安装的:

```
ansible 192.168.1.206 -m shell -a 'yum list installed | grep nginx'
```

显示结果如下:

```
192.168.1.206 | success | rc=0 >>
nginx.x86_64                1.8.0-1.el6.ngx @nginx
```

其实也可以用命令查看 yum 模块的帮助文件, 它本身就提供了强大的案例参考, 命令如下:

```
ansible-doc yum
```

显示结果如下所示 (显示结果较多, 只摘取了 Examples 部分的内容):

EXAMPLES:

- name: install the latest version of Apache
yum: name=httpd state=latest
- name: remove the Apache package
yum: name=httpd state=absent
- name: install the latest version of Apache from the testing repo


```

yum: name=httpd enablerepo=testing state=present
- name: install one specific version of Apache
  yum: name=httpd-2.2.29-1.4.amzn1 state=present
- name: upgrade all packages
  yum: name=* state=latest

- name: install the nginx rpm from a remote repo
  yum: name=http://nginx.org/packages/centos/6/noarch/RPMS/nginx-release-
centos-6-0.el6.ngx.noarch.rpm state=present
- name: install nginx rpm from a local file
  yum: name=/usr/local/src/nginx-release-centos-6-0.el6.ngx.noarch.rpm
state=present
- name: install the 'Development tools' package group
  yum: name="@Development tools" state=present

```

12. cron 模块

cron 模块，顾名思义就是创建计划任务，可以定义 webserver 组被控端机器每天凌晨 1 点过 1 分 ntpdate 自动对时，命令如下所示：

```

ansible webserver -m cron -a "name=ntpdate time every day" minute="1" hour="1"
job="/sbin/ntpdate ntp.api.bz >> /dev/null"

```

这里定义的 name 是标记计划任务，可以通过此标记删除或更改计划任务，命令显示结果如下：

```

192.168.1.205 | success >> {
  "changed": true,
  "jobs": [
    "ntpdate time every day"
  ]
}

```

```

192.168.1.206 | success >> {
  "changed": true,
  "jobs": [
    "ntpdate time every day"
  ]
}

```

详细语法可以参考 [ansible-doc cron](#)，这里不再重复命令显示结果。

13. service 模块

被控端服务管理，例如开启、关闭、重启服务等。

示例一：在 webserver 端开启 Nginx 服务，命令如下：

```

ansible webserver -m service -a "name=nginx state=started"

```

示例二：将 httpd 服务加入 webserver 端的启动项，命令如下：

```

ansible mysql -m service -a 'name=mysql state=started enabled=yes'

```

其他常用模块，在这里就不一一列举了，大家可以参考 Ansible 官方文档。

4.5 playbook 介绍

playbook 是一个不同于 Ansible 命令行执行方式的模式，其功能更为强大灵活。简而言之，它是一个非常简单的配置管理和多主机部署系统。playbook 是由一个或多个 “play” 组成的列表。play 的主要功能在于将事先归为一组的主机装扮成通过 Ansible 中的 task 事先定义好的角色。从根本上来讲，所谓的 task 就是调用 Ansible 的一个个模块将多个 play 组织在一个 playbook 中，这样就可以让它们联通起来按事先编排的机制同唱一台大戏。

playbook 的模板是使用 Python 的 jinja2 模块来处理的。学习过 Saltstack 的朋友对此模板应该都是比较熟悉的。另外，playbook 也是通过 YAML 格式来进行描述定义的，可以实现多台主机的应用部署，语法也并不复杂，大家可以对应官方案例学习其语法，官方网站提供了大量的案例，其地址为 <https://github.com/ansible/ansible-examples>。

下面先来看一下 Ansible 官方网站的一个案例，以举例说明 playbook 的用法。

```
#选择的主机组
- hosts: webserver

#定义的变量
vars:
    user: www
    group: www
    maxclients: 2000
    DocumentRoot: /var/www/html

#远端的执行权限
remote_user: root

#task是定义任务列表
tasks:
    #利用yum模块来操作
    - name: ensure apache is at the latest version
    #建议每个任务事件都要定义一个name标签，这样做既增强了可读性，又便于观察结果输出。
      yum: pkg=httpd state=latest
    - name: Apache Config File
      template: src=/home/yhc/apache/httpd.conf.j2 dest=/etc/httpd/conf/httpd.conf
      #src为Ansible主控端模块存放位置，dest为被控端httpd配置文件位置
    #触发重启服务器
      notify:
        - restart apache
    - name: ensure apache is running
      service: name=httpd state=started

#这里的restart apache 和上面的触发是配对的，这就是handlers（处理程序）的作用。
handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

模板文件 /home/yhc/apache/httpd.conf.j2 可参考官方案例，建议以 j2 结尾，表明这是

一个经过 jinja2 模板渲染的文件，并且存放在名为 templates 的子目录中。

定义的变量最好跟模板文件 /home/yhc/apache/httpd.conf.j2 中的变量一一对应，不然后面执行 ansible-playbook 时会报错，由于模板文件内容太长，这里只摘录跟变量相对应的内容，如下：

```
User {{ user}}
Group {{ group}}
DocumentRoot "{{DocumentRoot}}"
MaxClients      {{maxclients}}
```

语法简单明了，YAML 文件中的变量以“{{ 变量名 }}"表示，该文件若写得过于复杂，就会有语法错误的问题，可以采用如下方式检查语法错误：

```
ansible-playbook /home/yhc/httpd.yml --list-hosts --list-tasks
playbook: /home/yhc/httpd.yml
  play #1 (webserver): host count=2
    192.168.1.205
    192.168.1.206
  play #1 (webserver): TAGS: []
    ensure apache is at the latest version TAGS: []
    write the apache config file TAGS: []
    ensure apache is running TAGS: []
```

执行我们预先写好的 YAML 文件，路径为 /home/yhc/httpd.yml，命令如下：

```
ansible-book /home/yhc/httpd.yml -f 10
```

显示结果如下所示：

```
PLAY [webserver] *****
GATHERING FACTS *****
ok: [192.168.1.205]
ok: [192.168.1.206]
TASK: [ensure apache is at the latest version] *****
ok: [192.168.1.206]
ok: [192.168.1.205]
TASK: [write the apache config file] *****
changed: [192.168.1.205]
changed: [192.168.1.206]
TASK: [ensure apache is running] *****
ok: [192.168.1.205]
ok: [192.168.1.206]
NOTIFIED: [restart apache] *****
changed: [192.168.1.205]
changed: [192.168.1.206]
PLAY RECAP *****
192.168.1.205          : ok=5    changed=2    unreachable=0    failed=0
192.168.1.206          : ok=5    changed=2    unreachable=0    failed=0
```

ansible-playbook 后面紧跟着的就是我们所写的 /home/yhc/httpd.yml 文件，它的默认并

行进程数为 5，可以带上参数 `-f 10` 或更大的数值以提高并行进程数。

playbook 文件的详细说明

(1) 定义主机和用户

每份 playbook 文件都需要指定针对哪些主机进行运维，而 `hosts` 变量则说明了这个问题，`users` 则说明了采用哪个用户执行这条命令。

针对 `webserver` 主机组，这里采用 `root` 用户执行命令，代码如下：

```
---
- hosts: webservers
  remote_user: root
```

如果是 AWS EC2 主机，可以采用 `sudo` 模式执行命令，代码如下：

```
---
- hosts: webservers
  remote_user: ec2-user
  sudo: yes
```

(2) 任务列表

每一个 playbook 都会有一份任务列表（tasks list），说明究竟要按照怎样的顺序去执行这些命令（从上至下，依照顺序执行 task）。

使用 `service` 模块的命令如下：

```
tasks:
- name: make sure apache is running
  service: name=httpd state=running
```

使用 `command` 模块的命令如下：

```
tasks:
- name: disable selinux
  command: /sbin/setenforce 0
```

使用 `shell` 模块的命令如下：

```
tasks:
- name: run this command and ignore the result
  shell: /usr/bin/somecommand || /bin/true
```

使用 `copy` 模块的命令如下：

```
tasks:
- name: Copy ansible inventory file to client
  copy: src=/etc/ansible/hosts dest=/etc/ansible/hosts
       owner=root group=root mode=0644
```

使用 `template` 模块的命令如下：

```
tasks:
```

```
- name: create a virtual host file for {{ vhost }}
  template: src=somefile.j2 dest=/etc/httpd/conf.d/{{ vhost }}
```

(3) handlers

当被控端主机配置文件发生变化以后，通知处理程序 **handlers** 来触发后续的动作，比如重启 Apache 服务。在没有通知触发时 **handlers** 中定义的处理程序是不会执行的，触发是通过 **handlers** 定义的 **name** 标签来识别的，比如下面的 **notify** 中的 “**restart apache**” 和 **handlers** 中的 “**name: restart apache**” 内容请保持一致。

```
notify:
  - restart apache
  - name: ensure apache is running
    service: name=httpd state=started
handlers:
  - name: restart apache
    service: name=httpd state=restarted
```

下面简单介绍下 **playbook** 的条件语句与循环，语法非常简单，直接通过示例即可说明。条件语句 **when** 的示例如下：

```
tasks:
  - name: reboot redhat host
    command: /usr/sbin/reboot
    when: ansible_os_family == "RedHat"
```

循环语句的示例如下：

```
tasks:
  - name: install LNMP
    yum: name={{ item }} state=present
    with_items:
      - nginx
      - mysql-server
      - php-fpm
```

循环还支持列表，使用的是 **with_flattened** 语句。

变量文件的示例如下：

```
---
packages_LNMP:
  - [ 'nginx', 'mysql-server', 'php-fpm' ]
引用
- name: Install LNMP
  yum: name={{ item }} state=present
  with_flattened:
    - packages_LNMP
```

关于 **playbook** 文件的更多说明可参考文档 http://docs.ansible.com/playbooks_roles.html。

工作中经常会有这样一个需求：被控机上有 3 个用户（**yhc**、**admin** 和 **readonly**），分别

对应 3 套公私钥（分别对应不同的权限），需要用 Ansible 主控端进行公钥推送。这时，ssh-copy-id.yml 文件如下所示：

```
# Using alternate directory locations:
- hosts: webserver
  user: root
  tasks:
    - name: ensure users is present
      user: name={{ item }} state=present
      with_items:
        - yhc
        - admin
        - readonly

    - name: ssh-copy-id user yhc
      authorized_key: user=yhc key='{{ lookup('file', '/home/yhc/ansible/ssh-copy-id/example-master.pub') }}'

    - name: ssh-copy-id user admin
      authorized_key: user=admin key='{{ lookup('file', '/home/yhc/ansible/ssh-copy-id/example-operation.pub') }}'

    - name: ssh-copy-id user readonly
      authorized_key: user=readonly key='{{ lookup('file', '/home/yhc/ansible/ssh-copy-id/example-readonly.pub') }}'
```

authorized_key 是 Ansible 官方新出的一个模块，作用为添加或删除用户 SSH 公钥（adds or removes an SSH authorized key），这里主要用于添加用户公钥，详细说明请参见：http://docs.ansible.com/ansible/authorized_key_module.html。

如果 Ansible 部署在 AWS EC2 主机上，则默认是不允许 root 进行 SSH 的，并且 root 不提供密码，因此这里需要用有一个有 sudo 权限的用户来执行，默认用户一般是 ec2-user。

需要注意的是，这里的公钥文件全部都在 Ansible 主控机器的 /home/yhc/ansible/ssh-copy-id 目录下，而且不必担心被控机器端的 .ssh 目录是否建立、authorized 文件权限是否为 600 等，这些全部由 authorized_key 模块全自动完成，是不是很人性化呢？

此时的 ssh-copy-id 文件调整如下：

```
- hosts: bidder
  user: ec2-user
  sudo: yes
  tasks:
    - name: ensure users is present
      user: name={{ item }} state=present
      with_items:
        - bilin
        - readonly
      sudo: yes

    - name: ssh-copy-id user bilin
```

```
authorized_key: user=bilin key='{{ lookup('file', '/home/yhc/ansible/ssh-copy-id/example-operation.pub') }}'
- name: ssh-copy-id user readonly
  authorized_key: user=readonly key='{{ lookup('file', '/home/yhc/ansible/ssh-copy-id/example-readonly.pub') }}'
```

可用如下命令执行 ssh-copy-id.yml 文件，如下所示：

```
ansible-playbook -i hosts ssh-copy-id.yml
```

在被控端机器上进行检查，可发现公钥都已经正确分发了，而且权限自动地分配成了 600 权限，在主控端上切换相应的用户，SSH 登录也是正常的，说明整个配置过程是没有问题的。

4.6 角色

Ansible 的角色（roles）是 1.2 版本引入的新特性，用于层次性结构地组织 playbook。角色能够根据层次性结构自动装载 vars 变量文件、tasks 及 handlers 等。下面将采用角色来差异性地配置 webserver 组的 Nginx 服务，被控端机器具体配置信息如表 4-1 所示。

表 4-1 被控端机器详细配置表

被控端 IP	主机名	组名	CPU 核数
192.168.1.205	client1.example.com	webserver	4
192.168.1.206	client2.example.com	webserver	2

这里要注意区别一下，被控端主机 205 的 CPU 核数为 4，被控端主机 206 的 CPU 核数为 2。下面将利用 Ansible 的角色功能差异化地配置被控端的 Nginx 配置文件。这里是将配置文件放置在 /home/yhc/ansible/nginx 目录下，其目录结构如下：

```
nginx
├── hosts
├── roles
│   ├── common
│   │   ├── files
│   │   │   ├── epel-release-6-8.noarch.rpm
│   │   │   └── epel.repo
│   │   ├── handlers
│   │   └── tasks
│   │       └── main.yml
│   └── nginx
│       ├── handlers
│       │   └── main.yml
│       ├── tasks
│       │   └── main.yml
│       ├── templates
│       │   └── nginx.conf.j2
└── site.yml
```

其中,

- **site.yml 文件**: 为全局配置文件, 一般来说, 由此文件来引用角色, 通过 `hosts` 参数来绑定角色对应的主机或组。
- **hosts 文件**: 非必选配置, 用来指定主机或组, 默认将引用 `/etc/ansible/hosts` 文件, 通过 `-i` 参数来调用, 例如: `ansible-playbook -i hosts`。
- **common 角色目录**: 此外添加了一个公共类角色 `common`, 一般作用于被控端机器, 主要用于系统的基础服务, 例如添加 `epel` 源、`ntpdate` 自动对时、`sysctl` 内核优化等。
- **nginx 目录**: 用于 `Nginx` 角色目录。
- **files 目录**: 存放有 `copy` 或 `script` 等模块调用的文件。
- **vars 目录**: 定义 `playbook` 运行时需要使用的变量。
- **templates 目录**: `template` 模块会自动在此目录中寻找 `jinja2` 模板文件并渲染。
- **handlers 目录**: 此目录中应当包含一个 `main.yml` 文件, 用于定义各角色用到的各个 `handlers` 动作。
- **tasks 目录**: 此目录中至少要包含一个名为 `main.yml` 的文件, 用于定义此角色的任务列表, 可使用 `include` 指令。

1. site.yml 文件

`site.yml` 文件内容如下:

```
---
- name: configure and deploy the webserver
  hosts: webserver
  roles:
  - common
  - nginx
```

2. hosts 文件

`hosts` 文件内容如下所示:

```
[webserver]
192.168.1.205
192.168.1.206
```

语法和内容基本跟 `/etc/ansible/hosts` 一样, 这里就不再详细描述了。

3. common 角色目录

`common` 角色目录对应了 3 个子目录: `files`、`tasks` 和 `handles` 目录。

`files` 目录下有 `epel.repo` 文件, 方便利用 `copy` 模块推送至各控制端机器, 因为 `CentOS` 官方源并没有提供 `Nginx` 的安装, 所以这里采用 `epel` 进行安装, `epel.repo` 文件内容如下所示:

```
[epel]
```

```

name=Extra Packages for Enterprise Linux 6 - $basearch
baseurl=http://download.fedoraproject.org/pub/epel/6/$basearch
#mirrorlist=https://mirrors.fedoraproject.org/metalink?repo=epel-6&arch=$basearch
failovermethod=priority
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-6

[epel-debuginfo]
name=Extra Packages for Enterprise Linux 6 - $basearch - Debug
#baseurl=http://download.fedoraproject.org/pub/epel/6/$basearch/debug
mirrorlist=https://mirrors.fedoraproject.org/metalink?repo=epel-debug-6&arch=$basearch
failovermethod=priority
enabled=0
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-6
gpgcheck=1

[epel-source]
name=Extra Packages for Enterprise Linux 6 - $basearch - Source
#baseurl=http://download.fedoraproject.org/pub/epel/6/SRPMS
mirrorlist=https://mirrors.fedoraproject.org/metalink?repo=epel-source-6&arch=$basearch
failovermethod=priority
enabled=0
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-6
gpgcheck=1

```

tasks 目录下有 main.yml 文件，内容如下：

```

---
- name: Copy the EPEL repository definition
  copy: src=epel.repo dest=/etc/yum.repos.d/epel.repo

- name: Create the GPG key for EPEL
  command: rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-6

```

handlers 目录目前无文件，是空目录，因为执行的 copy 和 command 命令无须 handlers 启动服务或重启机器，所以此目录暂时为空。虽然是空目录，但建议保留。

4. Nginx 角色目录

Nginx 角色目录对应 3 个子目录：tasks、templates 和 handlers 目录。templates 目录中 nginx.conf.j2 文件内容如下：

```

user          nginx;
worker_processes {{ ansible_processor_cores }};
{% if ansible_processor_cores == 2 %}
worker_cpu_affinity 01 10;
{% elif ansible_processor_cores == 4 %}
worker_cpu_affinity 1000 0100 0010 0001;
{% elif ansible_processor_cores >= 8 %}
worker_cpu_affinity 00000001 00000010 00000100 00001000 00010000 00100000

```

```

01000000 10000000;
{% else %}
worker_cpu_affinity 1000 0100 0010 0001;
{% endif %}
worker_rlimit_nofile 65535;
events {
    use epoll;
    worker_connections 51200;
}
http {
    include      /etc/nginx/mime.types;
    default_type application/octet-stream;
    log_format   main '$remote_addr - $remote_user [$time_local] "$request" '
        '$status $body_bytes_sent "$http_referer" '
        '"$http_user_agent" "$http_x_forwarded_for"';
    access_log   /var/log/nginx/access.log main;
    sendfile     on;
    #tcp_nopush  on;
    keepalive_timeout 65;
    #gzip        on;
    include      /etc/nginx/conf.d/*.conf;
}

```

在这个文件中，`ansible_processor_cores` 变量是通过 Facts 组件获取到的，它在 Ansible 中是非常有用的组件，用于获取被控端主机的系统信息，包括主机名、操作系统、分区信息、硬件信息等，所以能够轻易地获取 CPU 核数，也可以通过运行 `ansible 192.168.1.206 -m setup` 命令来获取 206 被控端机器的完整 Facts 信息，命令显示的部分结果如下（因内容过多，这里只截取部分内容）：

```

192.168.1.206 | success >> {
  "ansible_facts": {
    "ansible_all_ipv4_addresses": [
      "192.168.1.206"
    ],
    "ansible_all_ipv6_addresses": [
      "fe80::216:3eff:fe08:ea2b"
    ],
    "ansible_architecture": "x86_64",
    "ansible_bios_date": "",
    "ansible_bios_version": "",
    "ansible_cmdline": {
      "KEYTABLE": "us",
      "LANG": "en_US.UTF-8",
      "SYSFONT": "latarcyrheb-sun16",
      "console": "hvc0",
      "quiet": true,
      "rd_LVM_LV": "VolGroup/lv_root",
      "rd_NO_DM": true,
      "rd_NO_LUKS": true,

```



```

"rd_NO_MD": true,
"rhgb": true,
"ro": true,
"root": "/dev/mapper/VolGroup-lv_root"
},
"ansible_date_time": {
  "date": "2015-11-29",
  "day": "29",
  "epoch": "1448799399",
  "hour": "12",
  "iso8601": "2015-11-29T12:16:39Z",
  "iso8601_micro": "2015-11-29T12:16:39.648209Z",
  "minute": "16",
  "month": "11",
  "second": "39",
  "time": "12:16:39",
  "tz": "UTC",
  "tz_offset": "+0000",
  "weekday": "Sunday",
  "year": "2015"
},

```

我们可以通过管道符命令来获取所需要的 Facts 信息，例如 CPU 核数，命令如下所示：

```
ansible 192.168.1.206 -m setup | grep ansible_processor_cores
```

命令显示结果如下所示：

```
ansible_processor_cores": 2,
```

还可以通过此命令来获取被控端 FQDN 完整名，并将其作为 Apache 配置文件中的 ServerName 参数值，命令如下所示：

```
"ansible_fqdn": "client2.example.com",
```

tasks 目录中的 main.yml 文件内容如下所示：

```

---
- name: ensure nginx is thd lastest version
  yum: name=nginx state=lastest

- name: Copy nginx configuration
  template: src=nginx.conf dest=/etc/nginx/nginx.conf
  notify: restart nginx

- name: ensure nginx is running
  service: name=nginx state=started

```

handlers 目录中的 main.yml 文件内容如下：

```

- name: restart nginx
  service: name=nginx state=restarted

```

运行角色，命令如下：

```
cd /home/yhc/ansible/nginx
ansible-playbook -I hosts site.yml
```

命令显示结果如下所示:

```
PLAY [webserver] *****
GATHERING FACTS *****
ok: [192.168.1.205]
ok: [192.168.1.206]
TASK: [common | Copy the EPEL repository definition] *****
ok: [192.168.1.205]
ok: [192.168.1.206]
TASK: [common | Create the GPG key for EPEL] *****
changed: [192.168.1.205]
changed: [192.168.1.206]
TASK: [nginx | ensure nginx is thd latest version] *****
changed: [192.168.1.206]
changed: [192.168.1.205]
TASK: [nginx | Copy nginx configuration] *****
changed: [192.168.1.206]
changed: [192.168.1.205]
TASK: [nginx | ensure nginx is running] *****
changed: [192.168.1.206]
changed: [192.168.1.205]
NOTIFIED: [nginx | restart nginx] *****
changed: [192.168.1.206]
changed: [192.168.1.205]
PLAY RECAP *****
192.168.1.205      : ok=7    changed=5    unreachable=0    failed=0
192.168.1.206      : ok=7    changed=5    unreachable=0    failed=0
```

现在来检查下 webserver 组两台机器的 Nginx 配置文件, 命令如下:

```
ansible webserver -m command -a 'cat /etc/nginx/nginx.conf'
```

如果命令结果如下, 则表示配置是成功的:

```
192.168.1.206 | success | rc=0 >>
user          nginx;
worker_processes 2;
worker_cpu_affinity 01 10;
```

```
worker_rlimit_nofile 65535;
events {
```

```
    use epoll;
```

```
    worker_connections 51200;
```

```
}
```

```
http {
```

```
    include          /etc/nginx/mime.types;
```

```
    default_type     application/octet-stream;
```

```
    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
```

```

        '$_status $_body_bytes_sent "$http_referer" '
        '"$http_user_agent" "$http_x_forwarded_for"';
    access_log /var/log/nginx/access.log main;
    sendfile      on;
    #tcp_nopush    on;
    keepalive_timeout 65;
    #gzip on;
    include /etc/nginx/conf.d/*.conf;
}

192.168.1.205 | success | rc=0 >>
user      nginx;
worker_processes 4;

worker_cpu_affinity 1000 0100 0010 0001;

worker_rlimit_nofile 65535;
events {
    use epoll;
    worker_connections 51200;
}
http {
    include      /etc/nginx/mime.types;
    default_type application/octet-stream;
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
        '$_status $_body_bytes_sent "$http_referer" '
        '"$http_user_agent" "$http_x_forwarded_for"';
    access_log /var/log/nginx/access.log main;
    sendfile      on;
    #tcp_nopush    on;
    keepalive_timeout 65;
    #gzip on;
    include /etc/nginx/conf.d/*.conf;
}

```

4.7 Jinja2 过滤器

这里补充一个重要的概念——Jinja2 过滤器，希望大家能够掌握。这是因为 Ansible 除了使用 YAML 文件以外，还大量使用 Jinja2 过滤器。

Jinja2 是 Python 下一个广泛应用的模板引擎，官网地址为 <http://jinja.pocoo.org>，下面来介绍下 Ansible 如何使用 Jinja2 的强大过滤器 (Filter) 功能。

1. 格式化数据

下面的过滤器会读取 template 中的数据结构并渲染为不同的格式，这点在调试的时候非常有用：

```
{{ 变量名 | to_json }}
```

```
{{ 变量名 | to_yaml }}
```

为了便于阅读, 可以使用:

```
{{ 变量名 | to_nice_json }}
{{ 变量名 | to_nice_yaml }}
```

从格式化数据读入的命令如下:

```
{{ 变量名 | from_json }}
{{ 变量名 | from_yaml }}
```

举例如下:

```
tasks:
- shell: cat /some/path/to/file.json
  register: result
- set_fact: myvar="{{ result.stdout | from_json }}"
```

和条件一起使用的示例如下:

```
tasks:
- shell: /usr/bin/foo
  register: result
  ignore_errors: True
- debug: msg="it failed"
  when: result|failed
- debug: msg="it changed"
  when: result|changed
- debug: msg="it succeeded"
  when: result|success
- debug: msg="it was skipped"
  when: result|skipped
```

2. 强定义变量

对于未定义变量, Ansible 的默认行为是 fail。但你可以将其关闭, 命令如下:

```
{{ 变量名 | mandatory }}
```

3. 未定义变量默认值

Jinja2 提供了一个有用的 default 过滤器, 相比于未定义变量时直接 fail, 这是个更好的方法:

```
{{ 变量名 | default(5) }}
```

4. 忽略未定义变量和参数

Ansible 1.8 之后, 可以使用 default 过滤器忽略未定义的变量和模块参数, 命令如下:

```
- name: touch files with an optional mode
  file: dest={{item.path}} state=touch mode={{item.mode|default(omit)}}
  with_items:
- path: /tmp/foo
```

```
- path: /tmp/bar
- path: /tmp/baz
mode: "0444"
```

5. list 过滤器

这些过滤器可作用在 list 的所有变量上。获取数字 list 中最小值的命令如下：

```
{{ list1 | min }}
```

获取数字 list 中最大值的命令如下：

```
{{ [3, 4, 2] | max }}
```

6. 集合过滤器

集合过滤器自带的函数都可以从集合或列表中返回一个唯一集合。

从 list 中获取唯一集合的命令如下：

```
{{ list1 | unique }}
```

两个 list 的并集、交集和差集，命令分别如下：

```
{{ list1 | union(list2) }}
{{ list1 | intersect(list2) }}
{{ list1 | difference(list2) }}
```

7. 随机数过滤器

从 list 中随机获取一个值，命令如下：

```
{{ ['a','b','c']|random }} => 'c'
```

从 0 到 59 中获取一个随机数，命令如下：

```
{{ 59 |random }}
```

从 0 到 100 以步长为 10 获取随机数，命令如下：

```
{{ 100 |random(step=10) }} => 70
```

从 1 到 100 以步长为 10 获取随机数，命令如下：

```
{{ 100 |random(1, 10) }} => 31
{{ 100 |random(start=1, step=10) }} => 51
```

8. shuffle 过滤器

该过滤器可随机排序已有 list，命令如下所示：

```
{{ ['a','b','c']|shuffle }} => ['c','a','b']
{{ ['a','b','c']|shuffle }} => ['b','c','a']
```

判断是否为数字，命令如下所示：

```
{{ myvar | isnan }}
```

求对数（默认基为 e）的命令如下所示：


```
{{ myvar | log }}
```

求 10 的对数，命令如下所示：

```
{{ myvar | log(10) }}
```

求次幂的命令如下所示：

```
{{ myvar | pow(2) }}
```

```
{{ myvar | pow(5) }}
```

求开方的命令如下所示：

```
{{ myvar | root }}
```

```
{{ myvar | root(5) }}
```

9. IP 过滤器

检查是否为有效 IP，命令如下所示：

```
{{ myvar | ipaddr }}
```

检查某版本是否有有效 IP，命令如下所示：

```
{{ myvar | ipv4 }}
```

```
{{ myvar | ipv6 }}
```

从 IP 地址提取指定信息，命令如下所示：

```
{{ '192.0.2.1/24' | ipaddr('address') }}
```

10. 哈希过滤器

使用哈希过滤器，命令如下所示：

```
{{ 'test1'|hash('sha1') }}
```

```
{{ 'test1'|hash('md5') }}
```

```
{{ 'test2'|checksum }}
```

```
{{ 'passwordsaresecret'|password_hash('sha512') }}
```

其他有用的过滤器用法如下所示：

获取路径的最后一个名称：{{ path | basename }}

从路径中获取目录名称：{{ path | dirname }}

获取链接的实际路径：{{ path | realpath }}

使用 **match** 或 **search** 匹配正则表达式的命令如下所示：

```
vars:
```

```
url: "http://example.com/users/foo/resources/bar"
```

```
tasks:
```

```
- shell: "msg='matched pattern 1'"
```

```
when: url | match("http://example.com/users/.*/resources/.*")
```

```
- debug: "msg='matched pattern 2'"
```

```
when: url | search("/users/.*/resources/.*")
```

使用 **regex_place** 进行正则替换的命令如下所示：

```
convert "ansible" to "able"
```

命令显示结果如下：

```
{{ 'ansible' | regex_replace('^a.*i(.*)$', 'a\\1') }}
```

使用 `convert` 进行正则替换，命令如下所示：

```
convert "foobar" to "bar"
```

命令显示结果如下：

```
{{ 'foobar' | regex_replace('^f.*o(.*)$', '\\1') }}
```

在正则中使用 `regex_escape` 转义特殊字符，命令如下所示：

```
convert '^f.*o(.*)$' to '\\^f\\.\\*o\\(\\.\\*\\)\\$'
```

命令显示结果如下：

```
{{ '^f.*o(.*)$' | regex_escape() }}
```

参考文献

http://docs.ansible.com/ansible/playbooks_filters.html#ip-address-filter。

4.8 小结

Ansible 跟自动化运维工具 Puppet（第 5 章会详细讲解）的关注方向不一样，它关注的是软件使用的便利性、简便性及扩展性，这也是很多运维人员和开发人员喜欢它的地方。另外，值得关注的是，AWS 最近的一份声明表示，Ansible 的多个模块还可以集成在 AWS 平台上，包括身份认证和访问管理功能等。用户可以在 Ansible 上创建基于 AWS 的自动化任务管理，包括用户、组员、角色管理，也可以进行相关规则设定。随着红帽公司收购了 Ansible，我们可以预见到 Ansible 会越来越成熟和流行。

自动化配置管理工具 Puppet

在大数据时代，高伸缩性、容错性、分布式的特点对系统运维提出了更高的要求，系统管理员不再疲于安装操作系统、对系统参数进行逐一配置与优化、打补丁、安装软件、配置软件、添加某个服务等操作了。他们开始做一些局部的自动化工作，这样可以提高效率、避免重复劳动、减少错误、积累知识。但这些还远远不够，要想满足大数据时代的运维需求，需要更彻底地应用自动化运维工具。

常见的运维工作流程包括：安装系统→优化系统与配置→安装软件→配置软件→添加监控→检查，等网站或系统正式上线后，后续可能还会有添加服务→配置变更→打补丁、修复漏洞等，是不是很烦琐？尤其是要负责部署及维护大量的服务器时，很多时候是无法凭借一己之力完成全部工作的，这时便需要一些自动化工具来帮忙了。

系统运维工作面临的各种不确定性问题更让人头疼，在 10 台 Linux 机器上变更应用相对来说还算很简单的事；就算到了 100 台，也有轻量级的 Fabric 自动化运维工具；但如果上升到 1000 台甚至上万台（尤其是其中还包含了数量不少的 Windows Server 机器）呢？那就会变得非常复杂了。而且重复性的劳动还会让人觉得疲惫和乏味，久而久之可能还会产生厌倦工作的情绪，如果这个时候使用自动化配置管理工具 Puppet，就能很轻松地解决这些问题了，这也是 Puppet 越来越流行的原因之一。

5.1 Puppet 的基本概念及介绍

5.1.1 Puppet 简介

Puppet 是一个基于 Ruby 开发的使用 GPL 协议授权的开源软件，它既能以客户端 - 服

务端的方式运行，也能独立运行；既可以用来管理 Linux、Unix 平台，也可以用来管理 Microsoft Windows。除此以外，它还可以用来管理一台主机的整个生命周期：从初始化、安装、升级、维护到最后将服务器迁移并下架，都可以用它来管理。Puppet 能够与主机持续进行交互，它并非是一个只负责搭建主机却不管理它们的工具。

5.1.2 学习 Puppet 应该掌握 Ruby 基础

在系统地学习和运用 Puppet 之前，推荐大家学习一下 Ruby 开发的基础知识，开发语言之间是相通的，之前有 Python 或 Java 开发基础的朋友可以多关注一下 Ruby 面向对象概念中的方法、类和模块，有兴趣的朋友也可以系统地学习一下 Ruby 语言，这里向大家推荐《Ruby 基础教程》第 4 版，本书为日本公认的最好的 Ruby 入门教程，松本行弘先生亲自审校并作序推荐。这本书支持最新的 Ruby 2.0，也附带讲解了可运行于 1.9 版本的代码，通俗易懂地讲解了编写程序时所需要的变量、常量、方法、类、流程控制等语法，以及主要类的使用方法和简单应用，让没有编程经验的读者也能轻松掌握 Ruby，找到属于自己的编程方式，做到融会贯通并且可灵活运用实际工作中。



说明 松本行弘先生是 Ruby 语言的发明者，亦是亚洲首屈一指的编程语言发明者。

5.1.3 Puppet 的基本概念及工作流程介绍

在使用任何软件之前都需要了解其工作原理，否则会给后续使用带来诸多不便。Puppet 采用了非常简单的服务器 - 客户端（即常用的 C/S）架构，所有数据的交互都通过 SSL 进行，以保证安全，它的工作模型示意图如图 5-1 所示。

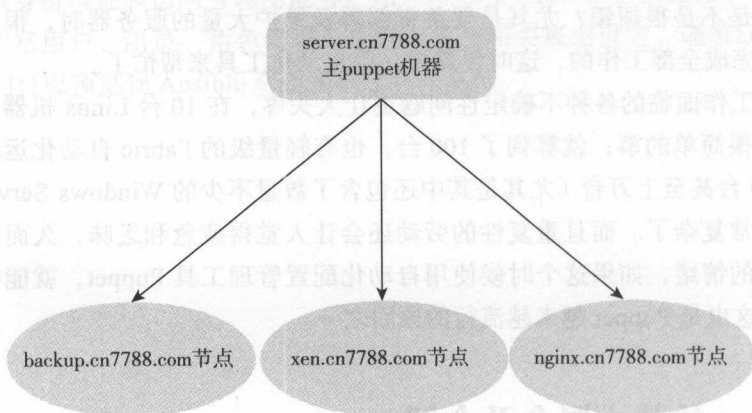


图 5-1 Puppet 的 C/S 模型图

通过图 5-1 可知，Puppet 会使用简单的 C/S 模型进行交互工作，服务端称为“Puppet Master 或 Puppet Server”，安装在客户端的软件称为 Puppet Agent。客户端主机本身被定义

为一个节点，如图 5-1 中的 backup.cn7788.com、xen.cn7788.com 及 nginx.cn7788.com。

1. 工作流程

Puppet 的工作流程如图 5-2 所示。

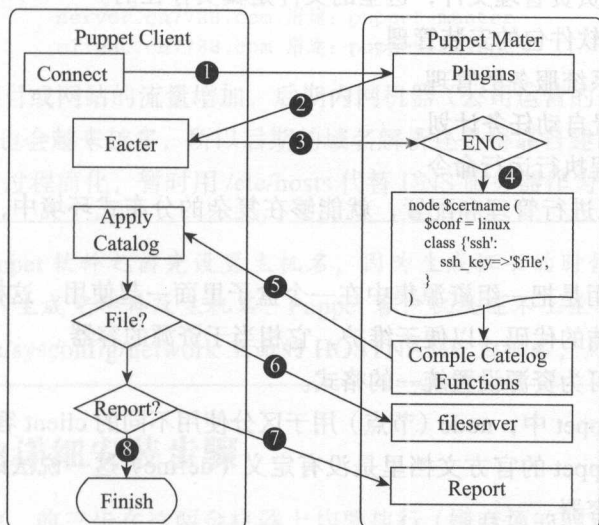


图 5-2 Puppet 工作流程图

下面具体说明 Puppet 的详细工作流程：

- 1) 客户端 Puppet-Client 向 Puppet-Master 端发起认证请求，或使用带签名的证书。
- 2) Puppet-Master 端告诉 Puppet-Client 是合法的。
- 3) Puppet-Client 调用 factor，factor 探测出主机的一些变量，例如主机名、内存大小、IP 地址等，Puppet-Client 将这些信息通过 SSL 连接发送到服务器端。
- 4) Puppet-Master 端检测客户端的主机名，然后找到 manifest 对应的 node 配置，并对该部分内容进行解析。factor 发送过来的信息将被当作变量处理，只有与 node 有牵涉的代码才会被解析，其他没牵涉的代码不解析。解析分为几个阶段，首先是语法检查，如果语法错误就报错；如果语法没错，就继续解析，解析的结果生成一个中间的“伪代码”（catalog），然后把伪代码发给客户端。
- 5) Puppet-Client 端接收到“伪代码”，并且执行。
- 6) Puppet-Client 端在执行时判断有没有 file 文件，如果有，则向 fileserver 发起请求。
- 7) Puppet-Client 端判断有没有配置 report，如果已配置，则把执行结果发送给服务器。
- 8) Puppet-Master 端把 Puppet-Client 端的执行结果写入日志，并发送给报告系统。

根据 Puppet 的工作流程可以发现，它跟我们之前用的 Ansible 推送模式不一样，它采取的是拉取模式，即安装在节点机器上的 Agent 程序定期向 Puppet Master（Server）服务器报备状态并拉取相应的配置信息。

2. Puppet 的基础概念

为了更好地理解 Puppet 的工作原理，下面介绍一下 Puppet 中经常见到的基础概念。

□ 资源：Puppet 涉及的资源主要有以下几种。

- file：主要负责管理文件，这里的文件是真实存在的。
- package：软件包的安装管理。
- service：系统服务的管理。
- cron：配置自动任务计划。
- exec：远程执行运行命令。

通过对这些资源进行管理和配置，就能够在复杂的分布式环境中，自动化地管理数据繁多的节点机器。

- 类：类的作用是把一组资源集中在一个盒子里面一起使用，这样做的好处是方便大家写出更简洁的代码，以便于维护，它相当于资源的容器。
- 模板：模板可为资源设置统一的格式。
- 节点：在 Puppet 中，node（节点）用于区分使用不同的 client 客户端。
- 定义：在 Puppet 的官方文档里是没有定义（define）这一说法的，可以将其理解为资源的组合容器。
- 模块：模块是资源、类、定义的组合，相当于更强的容器，模块名必须为普通字符，这里以名为 Nginx 的模块举例说明，它的路径位于 /etc/puppet/modules/nginx 下。一般来说，Nginx 模块会包含 3 个目录，分别为 manifests、files 及 templates 目录。
 - manifests 目录用于存放 init.pp 文件及其他配置文件，init.pp 文件是模块的核心，每一个模块都必须有一个 init.pp 文件。
 - files 目录用于存放模块目录需要用到的文件。
 - templates 目录则包含模块可能用到的模板文件。

5.2 安装 Puppet 前的准备工作

可以先准备 2 台 CentOS 6.4 x86_64 的机器，做好安装前的准备工作，这 2 台机器均要关闭 iptables 和 SELinux，另外，这 2 台机器都要先用 ntpdate 对时，这是因为 Puppet 的 C/S 两端进行同步时需要 SSL 验证，而 SSL 验证又依赖于主机上的正确时间，为了保证能向 Master 主机申请到正确的有效证书，建议先进行对时，命令如下：

```
ntpdate ntp.api.bz
```

以下结果表示此机器已成功对时。

```
31 Oct 03:17:06 ntpdate[988]: step time server 61.153.197.226 offset -184.108147 sec
```

通过 date 命令比对两边机器的时间，发现它们是一致的（上面的这些操作最好通过

Xshell4.0 的 “To All Session” 进行), 如下:

```
Sat Oct 31 03:17:10 UTC 2015
```

这 2 台内网机器的主机名和 IP 地址分配, 以及 /etc/hosts 文件内容分别如下:

```
192.168.1.124      server.cn7788.com 用途: puppet-master
192.168.1.125      client.cn7788.com 用途: puppet-client
```

当然了, 随着项目或网站的流量增加, 后期内网机器 (公司运营的网站也都是放在防火墙后面的局域网里) 也会越来越多, 所以后期的域名解析还是得靠自建的内部 DNS 服务器, 这里为了方便将演示过程简化, 暂时用 /etc/hosts 代替 DNS 服务器作为域名来进行解析。



注意 要在安装 Puppet 软件之前先设置主机名, 因为生成证书的时候要把主机名写入证书, 如果证书生成完毕再改主机名, Puppet 客户机就连不上主机了, 所以大家记得要先更改 /etc/sysconfig/network 里面的 HOSTNAME 主机名, 然后还要重启机器。

5.3 Puppet 的详细安装步骤

在下面的步骤中, 前三步在这两台机器上均要执行 (嫌麻烦的朋友可以利用 Xshell4.0 的 “To All Session” 功能进行批量部署)。

1) Puppet 不在 CentOS 的基本源中, 需要加入 PuppetLabs 提供的官方源, 由于笔者的系统全是 CentOS 6.4 x86_64 系统, 所以这里采用的是适用于 64 位系统的 rpm 软件包, 命令如下:

```
cd /usr/local/src
wget http://yum.puppetlabs.com/el/6/products/x86_64/puppetlabs-release-6-7.noarch.rpm
rpm -ivh puppetlabs-release-6-7.noarch.rpm
```

2) 安装 Puppet 需要的软件包, 这里直接用 yum 进行安装, 命令如下:

```
yum install -y mysql mysql-devel mysql-server ruby ruby-devel ruby-irb ruby-mysql ruby-rdoc ruby-ri
```

大家都知道, Puppet 是基于 Ruby 进行开发的, 所以先关注一下 Ruby 的版本, 命令如下:

```
ruby -v
```

命令显示结果如下所示:

```
ruby 1.8.7 (2013-06-27 patchlevel 374) [x86_64-linux]
```

3) 安装 Puppet 之前需要先安装 facter, 它的作用是收集主机的一些资料, 比如 CPU、主机 IP 等, facter 把收集到的值发送给 Puppet 服务器端, 服务器端就可以根据不同的条件来对不同的节点机器生成不同的 Puppet 配置文件。值得注意的是, Puppet-2.6.3 这个版本有

Bug, 在配置 `fileservers.conf` 文件进行文件推送时, 修改此文件会直接导致 `puppetmaster` 进程死掉, 所以建议大家安装时略过此版本, 选择更高级更稳定的 Puppet 版本。这里直接安装的是 3.8.3 版本, 可以用如下命令查看其版本号:

```
puppet --version
```

结果显示如下所示:

```
3.8.3
```



注意 如果 Puppet-Master 与 Puppet-Client 版本号不一致, 极有可能在 Puppet-Client 进行连接时产生 “ERROR 400 ON SERVER” 的报错, 所以建议大家尽量保持这两端的 Puppet 版本一致; 此外, 老版 Puppet 跟新版 Puppet 的许多命令都不一致, 请大家在阅读此章内容时注意这点。

下面是 Puppet 服务器端的配置步骤 (为了方便以后的操作和维护, 建议将 `puppetmaster` 配置成系统服务的形式来启动)。

1) 服务器端安装命令如下:

```
yum -y install puppet-server
```

2) 将 `puppetmaster` 服务配置成开机启动, 命令如下:

```
chkconfig puppetmaster on
```

3) 启动服务:

```
service puppetmaster start
```

4) 检查 `puppetmaster` 服务的启动情况。

第一次建议采用 `puppet master --verbose --no-daemonize` 命令启动, 这样有助于测试和调试错误, 如果采用命令方式启动, 可以看到启动的整个过程, 启动过程会做一些初始化的工作, 为 master 创建本地证书认证中心、证书和 key, 并打开 socket 等待 Puppet-Client 端的连接。可以在 `/var/lib/puppet/ssl` 目录看到相关的文件和目录, 命令显示结果如下所示:

```
Info: Creating a new SSL key for ca
Info: Creating a new SSL certificate request for ca
Info: Certificate Request fingerprint (SHA256): 43:F8:D3:D7:34:F8:C7:BA:DA:B5:0B:
25:BD:93:A1:93:20:5E:29:5F:CC:AE:1D:95:94:A0:60:C6:12:FC:12:2B
Notice: Signed certificate request for ca
Info: Creating a new certificate revocation list
Info: Creating a new SSL key for server
Info: csr_attributes file loading from /etc/puppet/csr_attributes.yaml
Info: Creating a new SSL certificate request for server
Info: Certificate Request fingerprint (SHA256): 5E:69:D3:D7:34:F8:C7:BA:DA:B5:0B:
03:0F:8D:DC:D3:E3:C6:07:09:FC:76:F3:50:29:DD:60:54:7A:5D:F5:19
Notice: server has a waiting certificate request
```

```

Notice: Signed certificate request for server
Notice: Removing file Puppet::SSL::CertificateRequest server at '/var/lib/puppet/
ssl/ca/requests/server.pem'
Notice: Removing file Puppet::SSL::CertificateRequest server at '/var/lib/puppet/
ssl/certificate_requests/server.pem'
Notice: Starting Puppet master version 3.8.3

```

此命令行的详细参数解释如下:

```

--no-daemonize  #前台输出日志
--verbose      #输入更加详细的日志
--debug        #还可以带上此参数,用于输出更加详细的日志,排错的时候使用

```

Puppet-Master 开启 Puppet 进程时占用的是 8140 端口, 可以用 lsof 命令来查看 8140 端口是否被占用, 命令如下所示:

```
lsof -i:8140
```

结果如下所示:

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE	NAME
puppet	1008	puppet	4u	IPv4	11158	0t0	TCP	*:8140 (LISTEN)

以上两条命令的结果说明 Puppet 服务器是正常启动的, puppet master --verbose --no-daemonize 和 service puppetmaster start 都是用于启动 Puppet-Server 端的, 不建议同时混用, 根据实际工作需求选其一即可。

讲完服务端的配置步骤, 现在来介绍一下 Puppet-Client 客户端的安装配置过程。前面的 yum 安装过程同 Puppet-Server 端一样, 这里不再重复, 请注意下面的过程跟 Puppet-Server 不一样, 要注意区分。

1) 客户端安装命令如下:

```
yum -y install puppet
```

2) 然后向 Puppet-Server 端发出请求, 命令如下:

```
puppet agent --test --server server.cn7788.com
```

此命令结果显示如下所示:

```

Info: Creating a new SSL key for client.cn7788.com
Info: Caching certificate for ca
Info: csr_attributes file loading from /etc/puppet/csr_attributes.yaml
Info: Creating a new SSL certificate request for client.cn7788.com
Info: Certificate Request fingerprint (SHA256): 60:08:6F:48:66:B9:B0:5D:7C:33:6E:
7D:04:DC:C3:CA:3B:9D:31:36:22:21:16:F0:47:19:08:CE:18:67:B6:E7
Info: Caching certificate for ca
Exiting; no certificate found and waitforcert is disabled

```

3) 此时在 Puppet-Server 端可以查看正在申请证书的客户端, 命令如下:

```
puppet cert --list -all
```


命令显示结果如下所示:

```
"client.cn7788.com" (SHA256) 60:08:6F:48:66:B9:B0:5D:7C:33:6E:7D:04:DC:C3:CA:3B:
  9D:31:36:22:21:16:F0:47:19:08:CE:18:67:B6:E7
+ "server" (SHA256) B1:57:D9:3B:55:FF:AA:70:3F:D8:BD:B3:1E:09:06:89:
  67:E1:20:CE:8E:84:E6:39:19:A0:E0:F4:6D:49:1A:66
```

此时要用如下命令接受请求:

```
puppet cert sign client.cn7788.com
```

命令结果显示信息如下:

```
Notice: Signed certificate request for client.cn7788.com
Notice: Removing file Puppet::SSL::CertificateRequest client.cn7788.com at '/var/
lib/puppet/ssl/ca/requests/client.cn7788.com.pem'
```

此命令表示接受客户机 **client.cn7788.com** 的认证。

4) Puppet-Client 端再发一次认证请求, 命令如下:

```
puppet agent --test --server server.cn7788.com
```

此命令结果显示如下所示:

```
Info: Caching certificate for client.cn7788.com
Info: Caching certificate_revocation_list for ca
Info: Caching certificate for client.cn7788.com
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Caching catalog for client.cn7788.com
Info: Applying configuration version '1446287484'
Info: Creating state file /var/lib/puppet/state/state.yaml
Notice: Finished catalog run in 0.07 seconds
```

此时 Puppet-Server 端显示结果如下所示:

```
Info: Caching node for client.cn7788.com
Info: Caching node for client.cn7788.com
Notice: Compiled catalog for client.cn7788.com in environment production in 0.12 seconds
```

Puppet-Client 端也可以采用服务的形式来启动, 命令如下:

```
Service puppet start
```

该启动方式跟 Puppet-Server 端的原理类似, 上面介绍的两种启动方式二选一即可, 后续的演示为了方便都是采用命令行的方式启动的。



曾有读者提出疑问, Puppet 的版本能够混用吗? 答案是可以, 但要注意的是 Puppet-Master 的版本一定要高于 Puppet-Client, 另外 Puppet-Master 和 Puppet-Client 之间的版本间隔不要相差太大, Puppet-Client 的版本越老, 与新版本的 Puppet-Master 一起正常运行的可能性也就越小。

5.4 Puppet 的简单文件应用

这里还是以上面两台机器为例进行说明。

在服务端 /etc/puppet/manifests/ 下建立文件 site.pp，此文件可以将 /tmp/andrew.txt 的内容和权限都推送过去，如果客户端存在此文件，则会采用此文件定义的文件内容和权限；如果不存在，则在推送过去之后，由所推送的文件来定义相应的内容和权限，文件内容如下：

```
node default{
  file {["/tmp/andrewy.txt":
    content =>"hello, My Name is Andrew.Yu !\n",
    ensure => present,
    mode => 644,
    owner => root,
    group => root,
  ]
}
```

Puppet 的基础结构是这样的：

```
类型 | 标题:
      属性=>值,
}
```

在上面的代码中，资源的类型是 file。Puppet 默认提供了很多资源类型，可以用来管理文件、服务、软件包及 cron 定时任务等。

文件中的其他内容都很好理解，只是 ensure=>present 表示什么意思呢？

ensure 后面可以接许多参数，如果后面接的是 present，则会检查该文件是否存在，如果不存在就新建该文件。

服务端先启动 puppetmaster 进程，命令如下：

```
service puppetmaster start
```

然后客户机 client.cn7788.com 执行如下命令：

```
puppet agent --test --server server.cn7788.com
```

命令显示结果如下：

```
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Caching catalog for client.cn7788.com
Info: Applying configuration version '1446519028'
Notice: /Stage[main]/Main/Node[default]/File[/tmp/yhc.txt]/ensure: created
Notice: Finished catalog run in 0.03 seconds
```

由上述结果可以看出 client.cn7788.com 中不存在 /tmp/yhc.txt 文件。

还可以再设定这样一种情况：client.cn7788.com 的节点机器存在 /tmp/yhc.txt 文件。演



示步骤如下所示：

1) 删除此文件后，在 /tmp 目录下再建立 yhc.txt 文件，文件内容如下：

```
Hi, This is a test file!
```

2) 然后在 client.cn7788.com 机器上再次执行连接 server.cn7788.com 端，命令如下：

```
puppet agent --test --server server.cn7788.com
```

此命令结果显示如下所示：

```
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Caching catalog for client.cn7788.com
Info: Applying configuration version '1446519028'
Notice: /Stage[main]/Main/Node[default]/File[/tmp/yhc.txt]/content:
--- /tmp/yhc.txt 2015-11-03 03:08:03.677046037 +0000
+++ /tmp/puppet-file20151103-5329-179slch-0 2015-11-03 03:08:18.765044005 +0000
@@ -1 +1 @@
-Hi,This is a test file!
+hello, My Name is yuhongchun !
Info: Computing checksum on file /tmp/yhc.txt
Info: /Stage[main]/Main/Node[default]/File[/tmp/yhc.txt]: Filebucketed /tmp/
yhc.txt to puppet with sum e38fd170e47129b097f377fd9bae116a
Notice: /Stage[main]/Main/Node[default]/File[/tmp/yhc.txt]/content: content
changed '{md5}e38fd170e47129b097f377fd9bae116a' to '{md5}448db1ed0c41f6da
c48d218e169463ae'
Notice: Finished catalog run in 0.08 seconds
```

从日志结果中就可以分析出来，yhc.txt 文件已经发生了变化。

上面的文件推送应用比较简单，我们应如何根据自己的实际需求，从 Puppet-Server 服务器端向 Puppet-Client 客户端分发指定的文件呢？比如说，要将服务器端的 /usr/local/src/softlist 文件集中分发到 Puppet-Client 的 /tmp 下面，应该如何操作呢？步骤如下。

1) 先修改 /etc/puppet/fileserver.conf 文件，命令如下：

```
[files]
path /usr/local/src
allow *
```



注意 files 在这里是一个虚拟目录，它实际对应的是服务器端的 /usr/local/src 目录，这里的语法跟 samba 服务器的语法是一样的，即定义的虚拟目录。

allow 后面连接的是允许连接到服务端主机地址的设置，这里设置为允许所有。

2) 再修改 /etc/puppet/manifests/ 下面的 site.pp 文件，内容如下：

```
file
{
  "/tmp/softlist":
    source => "puppet://server.cn7788.com/files/softlist",
    group => root,
}
```

```
owner => root,
mode => "644"
}
```

3) 在 client.cn7788.com 机器上执行如下命令(执行命令之前可以先修改 /etc/hosts 文件, 使之与 Puppet-Master 不一致):

```
puppet agent --test --server server.cn7788.com
```

命令显示结果如下:

```
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Caching catalog for client.cn7788.com
Info: Applying configuration version '1446535632'
Notice: /Stage[main]/Main/File[/usr/local/src/softlist]/content:
--- /usr/local/src/softlist      2015-11-03 06:46:28.994043464 +0000
+++ /tmp/puppet-file20151103-6644-mbchft-02015-11-03 07:27:15.504044004 +0000
@@ -1,4 +1,6 @@
-apache
-memcache
+nginx
+redis
-
+hadoop
+spark
+php-fpm
+pdns

Info: Computing checksum on file /usr/local/src/softlist
Info: /Stage[main]/Main/File[/usr/local/src/softlist]: Filebucketed /usr/local/
src/softlist to puppet with sum 7f43de0bcc4ac5f984f17775aba0ba96
Notice: /Stage[main]/Main/File[/usr/local/src/softlist]/content: content changed '{md
5}7f43de0bcc4ac5f984f17775aba0ba96' to '{md5}36fe51e7e690fe7d65046e9868ed2fa4'
Notice: /File[/usr/local/src/softlist]/seluser: seluser changed 'unconfined_u' to 'system_u'
Notice: Finished catalog run in 0.39 seconds
```

观察日志可以得知, Puppet-Client 的 /etc/hosts 文件已经更新了, 耗时 0.36 秒。

同理, 如果要推送 Puppet-Server 端的 /etc/crontab 文件, 只需要关注 Puppet-Server 端的这两个相关文件即可。

fileserver.conf 的文件内容如下:

```
[files]
path /etc/
allow *
```

/etc/puppet/manifests/site.pp 的文件内容如下:

```
file
{ "/etc/crontab":
source => "puppet://server.cn7788.com/files/crontab",
```

```
group => root,
owner  => root,
mode  => "644"
}
```

接着，让 client.cn7788.com 节点机器执行同步命令，命令如下：

```
puppetd --test --server server.cn7788.com
```

结果如下所示：

```
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Caching catalog for client.cn7788.com
Info: Applying configuration version '1446536063'
Notice: /Stage[main]/Main/File[/etc/crontab]/content:
--- /etc/crontab 2011-09-27 01:33:08.000000000 +0000
+++ /tmp/puppet-file20151103-6923-1twgwy5-0 2015-11-03 07:34:26.168044006 +0000
@@ -14,3 +14,10 @@
# | | | | |
# * * * * * user-name command to be executed

+00 01 * * * root /bin/bash /usr/local/nginx/sbin/cut_nginx_log.sh >> /dev/null 2>&1
+01 02 * * * root /bin/bash /root/backup.sh >> /dev/null 2>&1
+
+03 03 * * * root /bin/bash /root/sshdeny.sh >> /dev/null 2>&1
+#01 04 * * * root /bin/bash /root/rsync_dir.sh >>/dev/null 2>&1
+
+##*/5 * * * * root /etc/init.d/iptables stop
Info: Computing checksum on file /etc/crontab
Info: /Stage[main]/Main/File[/etc/crontab]: Filebucketed /etc/crontab to puppet
with sum 4f2aaa54c48dda350f75da151f79ae57
Notice: /Stage[main]/Main/File[/etc/crontab]/content: content changed '{md5}4f2aaa5
4c48dda350f75da151f79ae57' to '{md5}7e76ef490e02dde0dd8e82a5cf7c0c69'
Notice: Finished catalog run in 0.59 seconds
```

从日志可以得知，/etc/crontab 文件已经很顺利地推送过去了，client.cn7788.com 端的 /etc/crontab 文件已经跟 server.cn7788.com 端的 /etc/crontab 文件保持一致了，整个过程总共耗时 0.59 秒。

上面的例子都是分发文件，如果要强制推送文件夹呢？继续进行测试，编辑 /etc/puppet/manifests/site.pp 文件，文件内容如下：

```
file
{
  "/usr/local/src/test":
    source => "puppet://server.puppet.com/files/test",
    recurse => true,
    ensure => directory,
    force => true
}
```

其中的 `recurse => true` 是递归复制, `ensure => directory` 是确保客户端存在 `/usr/local/src/test` 目录, `force => true` 会强制删除或覆盖已存在的目录。

然后, 在客户端执行同步命令, 命令显示结果如下所示:

```
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Caching catalog for client.cn7788.com
Info: Applying configuration version '1446537314'
Notice: /Stage[main]/Main/File[/usr/local/src/test]/ensure: created
Notice: /Stage[main]/Main/File[/usr/local/src/test/yhc]/ensure: defined content
as '{md5}d41d8cd98f00b204e9800998ecf8427e'
Notice: /Stage[main]/Main/File[/usr/local/src/test/cc]/ensure: defined content as
'{md5}d41d8cd98f00b204e9800998ecf8427e'
Notice: /Stage[main]/Main/File[/usr/local/src/test/dd]/ensure: defined content as
'{md5}d41d8cd98f00b204e9800998ecf8427e'
Notice: Finished catalog run in 0.77 seconds
```

使用上面的配置继续测试会发现, 当服务端源文件夹内增加、更新文件时, 客户端会自动增加、更新相应的文件, 但在服务端源文件夹内删除文件时, 客户端不会自动删除, 如果工作中有同步删除需求, 可以考虑采用 Puppet 结合 `rsync` 的方式来实现。

`file` 资源是我们在使用 Puppet 时最常用的资源之一, 其配置方法也是多元化的, 而且是影响 Puppet 执行效率的关键。那么, 可以利用 `file` 资源做哪些工作呢?

- 管理文件内容、属性和权限等。
- 管理文件、目录、符号链接等。
- 通过属性来指定文件来源, 也可以通过 `source` 属性从远程服务器中下载。
- 设置 `recurse` 属性为 `true` 同步传输整个目录。

`file` 目前可使用的参数如下。

- `ensure` 参数: 这个参数指定是否创建、删除文件或目录, 有 `present`、`absent`、`file`、`directory` 等值。其中 `present` 会检查文件是否存在, 若不存在则会创建一个空文件; `absent` 会删除文件或目录, 如果是目录则需要通过 `recurse` 参数指定是否允许递归; 如果 `recurse` 参数指定的是其他的参数, 则会创建连接文件, 为了方便管理, 建议在创建的时候使用 `ensure => link`, 并通过 `target` 参数指定文件。
- `force` 参数: 该参数会强制执行删除文件、软链接和目录等相关操作, 进行清空子目录、修改文件或链接的目录、删除目录等操作时必须指定 `force` 参数, 并确保 `ensure = absent`。
- `group` 参数: 指定文件或目录的属组, 可以是组名或组 id, 如果是 Windows, 属组和属主不能相同。
- `ignore` 参数: 指定在递归期间对符合指定模式的文件所进行的操作将被忽略。
- `links` 参数: 指定处理文件期间如何处理链接文件, 可以设置为 `follow` 和 `manage`。在复制文件的时候, `follow` 将会复制目标文件来代替链接文件, `manage` 将只会复制

链接文件。

- ❑ **mode** 参数：用来指定文件或目录的权限，Puppet 使用的是传统的 Unix 权限方案。
- ❑ **owner** 参数：指定文件的属主，可以是用户名或用户 id，如果是 Windows，属组和属主不能相同。
- ❑ **path** 参数：指定文件管理的路径。
- ❑ **purge** 参数：用于删除在 Puppet-Server 上不存在的文件，这个参数只有在管理目录时指定了 `recurse => true` 参数才有意义。
- ❑ **recurse** 参数：指定是否进行递归调用，其值有 `true`、`false`、`inf` 和 `remote`。
- ❑ **source** 参数：指定将会被复制到指定位置的资源文件。
- ❑ **target** 参数：指定创建链接文件的目标文件或目录。

了解完 `file` 的参数，再来看看 `file` 资源的缺点。

`file` 资源不适合做大量文件的分发处理工作，如果文件数量增多了，效率就会下降，尤其是文件的属性比较多时。可以通过客户端的反馈结果发现，大量文件的分发处理工作是耗时最多的一个。在后面的内容中会提到，这个缺点可以结合 `rsync` 服务来弥补。

如果局域网内的机器非常多，每次从 Puppet-Client 向 Puppet-Server 端发送证书请求时都必须手动输入命令给客户端签名，那么在服务器上应该如何配置以确保自动安全地给客户端签名呢？步骤如下。

1) 编辑 `/etc/puppet/puppet.conf`，在 `[main]` 的最后添加如下内容：

```
autosign = true
```

2) 重启 `puppetmaster` 服务，命令如下：

```
service puppetmaster restart
```

3) 另外配置一台名为 `fabric.cn7788.com` 的 Puppet-Client 机器，安装过程此处略过，然后使用如下命令连接 Puppet-Server 机器：

```
puppet agent --test --server server.cn7788.com
```

此命令返回结果显示如下所示：


```
Info: Creating a new SSL key for fabric.cn7788.com
Info: Caching certificate for ca
Info: csr_attributes file loading from /etc/puppet/csr_attributes.yaml
Info: Creating a new SSL certificate request for fabric.cn7788.com
Info: Certificate Request fingerprint (SHA256): C4:91:56:7A:46:78:89:5E:DC:A1:B9:
93:23:DD:6D:31:82:AD:71:86:EC:86:D8:71:34:96:EE:4E:16:6A:33:80
Info: Caching certificate for fabric.cn7788.com
Info: Caching certificate_revocation_list for ca
Info: Caching certificate for ca
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Caching catalog for fabric.cn7788.com
```

```

Info: Applying configuration version '1446606289'
Notice: /Stage[main]/Main/File[/usr/local/src/test]/ensure: created
Notice: /Stage[main]/Main/File[/usr/local/src/test/cc]/ensure: defined content as '{md5}
d41d8cd98f00b204e9800998ecf8427e'
Notice: /Stage[main]/Main/File[/usr/local/src/test/dd]/ensure: defined content as '{md5}
d41d8cd98f00b204e9800998ecf8427e'
Info: Creating state file /var/lib/puppet/state/state.yaml
Notice: Finished catalog run in 0.43 seconds

```

通过日志分析，Puppet-Server 机器已经替名为 fabric.cn7788.com 的节点机器自动颁发了签证，表示上述配置是生效的。

 **注意** 自动颁发证书是一种比较危险的操作，除非有足够安全的保证，否则一般情况下不要这样操作。

另外，还有个问题：Puppet 客户端应该如何自动连接 Puppet-Master，并且如何修改默认连接时间间隔呢？

此时间间隔默认为 1800 秒（30 分钟），有时候在实际工作中需要更改此时间间隔，比如将其更改为 10 分钟一次，下面以 client.cn7788.com 的客户机为例说明具体的实现步骤。

首先，要修改它的 /etc/puppet/puppet.conf 文件，在 [puppet-client] 的最下面添加如下内容：

```


server=server.cn7788.com
runinterval=300

```

然后以服务的形式启动此 Puppet Agent，命令如下：

```
service puppet start
```

在上面的配置文件里，server 选项设置的是 Puppet-Server 的域名地址，在 Puppet-Client 端设置好此域名地址后，就可以以默认的 30 分钟为频率自动连接 Puppet-Server 了，在这里域名地址为 server.cn7788.com。其实配置了参数以后，Puppet-Client 客户端就可以自动向 Puppet-Master 发起连接了，而不再需要用 puppetd --test --server server.cn7788.com 命令进行手动连接。如果不指定此项参数，Puppet 客户机将只是单纯地启动 Puppetd 进程，而不会主动找 server.cn7788.com 发起请求。在实际应用中通常需要配置此选项，让客户端自动进行连接工作。runinterval 选项可用于设置每隔多长的时间进行一次自动更新，时间单位为秒，这里选择的是 300 秒。

 **注意** 一般来说，不要随意修改系统中默认配置的“30 分钟”这个值，特别是不能更改成一个过小的数值，这会导致在客户端数量比较大的工作场景中，Puppet-Server 因为响应不了频繁的连接而发生“timeout”的报错。

下面还是以 client.cn7788.com 节点机器为例来验证下，大家可以观察下 messages 系统日志，命令如下所示：

```
tail -f /var/log/messages
```

命令显示结果如下所示：

```
Nov 3 02:56:58 client puppet-agent[7882]: Applying configuration version '1446537314'
Nov 3 02:56:58 client puppet-agent[7882]: Finished catalog run in 0.38 seconds
Nov 3 22:11:28 client puppet-agent[8340]: Reopening log files
Nov 3 22:11:29 client puppet-agent[8340]: Starting Puppet client version 3.8.3
Nov 3 22:11:37 client puppet-agent[8343]: Finished catalog run in 0.32 seconds
Nov 3 22:12:57 client puppet-agent[8340]: Caught TERM; storing stop
Nov 3 22:12:58 client puppet-agent[8340]: Processing stop
Nov 3 22:13:00 client puppet-agent[8497]: Reopening log files
Nov 3 22:13:01 client puppet-agent[8497]: Starting Puppet client version 3.8.3
Nov 3 22:13:07 client puppet-agent[8500]: Finished catalog run in 0.10 seconds
Nov 3 22:18:07 client puppet-agent[8635]: Finished catalog run in 0.40 seconds
Nov 3 22:23:05 client puppet-agent[8766]: Finished catalog run in 0.10 seconds
Nov 3 22:28:06 client puppet-agent[8897]: Finished catalog run in 0.35 seconds
```

可以清楚地看到，Puppet-Client 每次自动连接的时间间隔都是 5 分钟，即 300 秒，从而验证了上面的配置是正确的。

5.5 Puppet 的进阶操作

这里的操作环境已按照前面的操作示例搭建完毕，如下：

```
server.cn7788.com 192.168.1.205 puppet-master
fabric.cn7788.com 192.168.1.204 puppet-client
client.cn7788.com 192.168.1.206 puppet-client
```

现在来清理前面的环境，清空 /etc/puppet/manifests/ 里的 site.pp 文件内容，并且通过 ntpdate 做好时间的精准对时，不然 Puppet-Client 连接时会因为时间的关系而连接不上。

5.5.1 如何同步 Puppet-Client 端上的常用服务

在同步 Puppet-Client 端上的常用服务之前，会有如下要求：

- ☐ 开启 httpd 服务和 postfix 服务。
- ☐ 关闭 vsftpd 服务。

同步的步骤其实并不复杂，还是在服务器端 /etc/puppet/manifests 的 site.pp 文件上进行操作，内容如下（为了实验方便，请提前用 yum 安装好下面的服务）：

```
service {
  ["httpd","postfix"]::
  ensure => running;
  "vsftpd":
```

```
ensure => stopped;
}
```

Puppet-Client 端顺利连接上服务器端后，以节点机器 client.cn7788.com 为例，正常显示结果应该如下所示：

```
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Caching catalog for client.cn7788.com
Info: Applying configuration version '1446608434'
Notice: /Stage[main]/Main/Service[postfix]/ensure: ensure changed 'stopped' to 'running'
Info: /Stage[main]/Main/Service[postfix]: Unscheduling refresh on Service[postfix]
Notice: Finished catalog run in 1.60 seconds
```

在客户端机上输入命令验证下实验结果：

```
service postfix status
master (pid 10028) is running...
service httpd status
httpd (pid 9603) is running...
service vsftpd status
vsftpd is stopped
```

结果表明需求已经正确完成了。

5.5.2 如何在 Puppet-Client 端自动安装常用的软件包

在 Puppet-Client 端自动安装 screen、ntp 及 sysstat 包的步骤如下。

首先还是编辑 /etc/puppet/manifests/site.pp 文件，内容如下：

```
package {
  ["screen", "ntp", "sysstat"]:
    ensure => "installed";
}
```

客户端顺利连接上服务器端后，以节点机器 client.cn7788.com 为例，正常显示结果应该如下所示：

```
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Caching catalog for client.cn7788.com
Info: Applying configuration version '1446608680'
Notice: /Stage[main]/Main/Package[sysstat]/ensure: created
Notice: Finished catalog run in 15.03 seconds
```

5.5.3 如何自动同步 Puppet-Client 端的 yum 源

Puppet-Master 机器上更新了 yum 源后，可通过以下命令观察：

```
ls ll /etc/yum.repos.d/
```


命令显示结果如下：

```
total 24
-rw-r--r--. 1 root root 1926 Feb 25 2013 CentOS-Base.repo
-rw-r--r--. 1 root root 638 Feb 25 2013 CentOS-Debuginfo.repo
-rw-r--r--. 1 root root 630 Feb 25 2013 CentOS-Media.repo
-rw-r--r--. 1 root root 3664 Feb 25 2013 CentOS-Vault.repo
-rw-r--r--. 1 root root 957 Nov 4 03:46 epel.repo
-rw-r--r--. 1 root root 0 Nov 4 03:46 nginx.repo
-rw-r--r--. 1 root root 1250 Apr 12 2013 puppetlabs.repo
```

如果想通过 Puppet-Master 端将这些文件都推送到 client.cn7788.com 及 fabric.cn7788.com 对应的目录 /etc/yum.repos.d/ 中去，该如何实现呢？步骤如下。

1) 修改 /etc/puppet/filesserver.conf 文件，内容如下：

```
[files]
path /etc/yum.repos.d/
allow *
```

2) 修改 /etc/puppet/manifests/site.pp 文件，内容如下：

```
file
{
  "/etc/yum.repos.d/":
    source => "puppet://server.cn7788.com/files/",
    group => root,
    owner => root,
    mode => 644,
    recurse => true,
    force => true,
    purge => true
}
```

客户端正确连接到 Puppet 服务器端后，大家可以发现，Puppet-Server 会向 Puppet-Client 端推送文件了，分别在两台节点机器上输入如下命令：

```
puppet agent --test --server server.cn7788.com
```

会看到两台机器的结果是一样的，输入命令的反馈结果如下：

```
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Caching catalog for client.cn7788.com
Info: Applying configuration version '1446619419'
Notice: /Stage[main]/Main/File[/etc/yum.repos.d/puppetlabs.repo]/ensure: defined
        content as '{md5}14d68f86efb69e928d626c7ea8a974b3'
Notice: /Stage[main]/Main/File[/etc/yum.repos.d/CentOS-Debuginfo.repo]/ensure:
        defined content as '{md5}8b95e819eaea42849932309b5be5533d'
Info: Computing checksum on file /etc/yum.repos.d/puppetlabs-release-6-7.noarch.rpm
Info: /Stage[main]/Main/File[/etc/yum.repos.d/puppetlabs-release-6-7.noarch.rpm]:
        Filebucketed /etc/yum.repos.d/puppetlabs-release-6-7.noarch.rpm to puppet
        with sum 2aa0affe57ade441bfb9dcd6126a6cd6
```



```

Notice: /Stage[main]/Main/File[/etc/yum.repos.d/puppetlabs-release-6-7.noarch.
rpm]/ensure: removed
Notice: /Stage[main]/Main/File[/etc/yum.repos.d/CentOS-Base.repo]/ensure: defined
content as '{md5}d03052aaa85e5d26451f0ada9054f50f'
Info: Computing checksum on file /etc/yum.repos.d/softlist
Info: /Stage[main]/Main/File[/etc/yum.repos.d/softlist]: Filebucketed /etc/yum.
repos.d/softlist to puppet with sum 36fe51e7e690fe7d65046e9868ed2fa4
Notice: /Stage[main]/Main/File[/etc/yum.repos.d/softlist]/ensure: removed
Info: /Stage[main]/Main/File[/etc/yum.repos.d/test]: Recursively backing up to filebucket
Info: Computing checksum on file /etc/yum.repos.d/test/dd
Info: /Stage[main]/Main/File[/etc/yum.repos.d/test]: Filebucketed /etc/yum.repos.d/
test/dd to puppet with sum d41d8cd98f00b204e9800998ecf8427e
Info: Computing checksum on file /etc/yum.repos.d/test/cc
Info: FileBucket got a duplicate file {md5}d41d8cd98f00b204e9800998ecf8427e
Info: /Stage[main]/Main/File[/etc/yum.repos.d/test]: Filebucketed /etc/yum.repos.
d/test/cc to puppet with sum d41d8cd98f00b204e9800998ecf8427e
Notice: /Stage[main]/Main/File[/etc/yum.repos.d/test]/ensure: removed
Notice: /Stage[main]/Main/File[/etc/yum.repos.d/nginx.repo]/ensure: defined
content as '{md5}d41d8cd98f00b204e9800998ecf8427e'
Notice: /Stage[main]/Main/File[/etc/yum.repos.d/CentOS-Media.repo]/ensure:
defined content as '{md5}db3010a594efc3043651d78741ac02ff'
Notice: /Stage[main]/Main/File[/etc/yum.repos.d/epel.repo]/ensure: defined
content as '{md5}e8950030d9c72cf3e7a6469e8c1404ca'
Notice: /Stage[main]/Main/File[/etc/yum.repos.d/CentOS-Vault.repo]/ensure: defined
content as '{md5}62b394965682a15877a36357fe55689d'
Notice: Finished catalog run in 0.75 seconds

```

同样，在 Puppet-Client 端用 `ll` 命令进行观察，可以发现客户端已经将 Puppet-Server 端 `/etc/yum.repos.d` 目录下的文件同步过来了，实现了此工作的需求，检查结果如下：

```

total 24
-rw-r--r--. 1 root root 1926 Nov  4 06:44 CentOS-Base.repo
-rw-r--r--. 1 root root  638 Nov  4 06:44 CentOS-Debuginfo.repo
-rw-r--r--. 1 root root  630 Nov  4 06:44 CentOS-Media.repo
-rw-r--r--. 1 root root 3664 Nov  4 06:44 CentOS-Vault.repo
-rw-r--r--. 1 root root  957 Nov  4 06:44 epel.repo
-rw-r--r--. 1 root root    0 Nov  4 06:44 nginx.repo
-rw-r--r--. 1 root root 1250 Nov  4 06:44 puppetlabs.repo

```

5.5.4 如何根据不同名字的节点机器推送不同的文件

如果需要让 Puppet-Server 端分别向名为 `client.cn7788.com` 的客户机推送 `/etc/crontab` 文件，向名为 `fabric.cn7788.com` 的客户机推送 `/etc/hosts` 文件，向名为 `nginx.cn7788.com` 的客户机推送 `/etc/resolv.conf` 文件，应该如何实现呢？

需求比较复杂，可以通过 Puppet 模块来实现需求，模块文件一般放置在服务器端的 `/etc/puppet/modules/` 下。在下面首先会定义一个名为 `pushfile` 的模块。模块是 Puppet 生态系统的核心部分，我们一般通过资源的定义告诉 Puppet 应该去做什么，通常会针对一个应用编写一个模块，例如即将定义的 `pushfile` 模块。

首先注释掉 `fileservers.conf` 文件里面的相关内容, 清空 `site.pp` 文件后输入新的内容, 然后在 `/etc/puppet/modules` 下面建立名为 `pushfile` 的模块, 操作如下:

```
mkdir -p /etc/puppet/modules/pushfile/{manifests,files,templates}
```

操作完成后, `site.pp` 文件的内容如下:

```
import "node.pp"
```

这里扩展了 `site.pp` 文件的内容, 它会载入 `node.pp` 文件, 这样 Puppet-Master 在启动的时候, 就会自动载入并处理 `node.pp` 文件。

这时, 服务器端的 `/etc/puppet/manifests/node.pp` 的文件内容如下:

```
node 'client.cn7788.com' {  
  file  
  {"/etc/crontab":  
    source => "puppet://server.cn7788.com/modules/pushfile/crontab",  
    group => root,  
    owner => root,  
    mode => 644,  
  }  
}  
  
node 'fabric.cn7788.com' {  
  file  
  {"/etc/hosts":  
    source => "puppet://server.cn7788.com/modules/pushfile/hosts",  
    group => root,  
    owner => root,  
    mode => 644,  
  }  
}  
  
node 'nginx.cn7788.com' {  
  file  
  {"/etc/resolv.conf":  
    source => "puppet://server.cn7788.com/modules/pushfile/resolv.conf",  
    group => root,  
    owner => root,  
    mode => 644,  
  }  
}
```

`node.pp` 的配置文件比较长并且也很复杂, 究竟应该使用什么方法来检查它的语法错误呢? 可以输入如下命令:

```
puppet parser validate node.pp
```

如果配置文件是正确的, 则什么也不显示; 如果检测到错误了, 则会以红色醒目字体来提示。

Puppet 是利用 node (节点) 来区分不同的客户端的, 它会给不同的客户端分配不同的资源。观察下 node.pp 文件, 在这个文件里, source 的值会告诉 Puppet 去哪里寻找文件, 这里将文件全部都置于 Puppet-Server 的 /etc/puppet/modules/site/files 目录下面了, 它相当于是根目录, 然后将服务器端的文件依次复制到此目录的 /etc 下, 操作命令如下:

```
cp /etc/{crontab,hosts,resolv.conf}
/etc/puppet/modules/pushfile/files
```

现在用 tree 命令来查看下 pushfile 模块的目录树结构, 如下:

```
tree /etc/puppet/modules/pushfile
```

命令显示结果如下所示:

```
/etc/puppet/modules/pushfile
├── files
│   ├── crontab
│   ├── hosts
│   └── resolv.conf
├── manifests
└── templates
```

3 directories, 3 files

下面依次在 3 台 Puppet-Client 机器上执行 puppet agent 命令, 以 client.cn7788.com 机器为例进行说明, 输入的命令如下:

```
puppet agent --test --server server.cn7788.com
```

命令显示结果如下所示:

```
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Caching catalog for client.cn7788.com
Info: Applying configuration version '1446623887'
Notice: /Stage[main]/Main/Node[client.cn7788.com]/File[/etc/crontab]/content:
--- /etc/crontab 2015-11-03 07:34:26.372044003 +0000
+++ /tmp/puppet-file20151104-17989-alsqkh6-0 2015-11-04 08:02:50.127072946 +0000
@@ -14,10 +14,9 @@
# | | | | |
# * * * * * user-name command to be executed

-00 01 * * * root /bin/bash /usr/local/nginx/sbin/cut_nginx_log.sh >> /dev/null 2>&1
+#00 01 * * * root /bin/bash /usr/local/nginx/sbin/cut_nginx_log.sh >> /dev/null 2>&1
01 02 * * * root /bin/bash /root/backup.sh >> /dev/null 2>&1

-03 03 * * * root /bin/bash /root/sshddeny.sh >> /dev/null 2>&1
-#01 04 * * * root /bin/bash /root/rsync_dir.sh >> /dev/null 2>&1
+#03 03 * * * root /bin/bash /root/sshddeny.sh >> /dev/null 2>&1


-#/5 * * * * root /etc/init.d/iptables stop
```

```

+*/5 * * * * root /etc/init.d/iptables stop
Info: Computing checksum on file /etc/crontab
Info: /Stage[main]/Main/Node[client.cn7788.com]/File[/etc/crontab]: Filebucketed
/etc/crontab to puppet with sum 7e76ef490e02dde0dd8e82a5cf7c0c69
Notice: /Stage[main]/Main/Node[client.cn7788.com]/File[/etc/crontab]/content:
content changed '{md5}7e76ef490e02dde0dd8e82a5cf7c0c69' to '{md5}2022061d798
f3933e0caafb614272212'
Notice: Finished catalog run in 0.47 seconds

```

可以看到配置是成功的，`/etc/crontab` 被成功地推送过来了，而且内容也进行了置换，其他节点机器的结果在这里就不打印了。

 **注意** 在 `/etc/puppet/modules` 下定义的模块都是自动载入的，所以不需要用 `import` 来加载。

5.5.5 如何根据节点机器名选择性地执行 Shell 程序

如果客户端机器 `nginx.cn7788.com` 没有安装 Nagios 客户端程序，想要通过 Puppet-Server 推送 Shell 脚本自动安装，其他节点机器暂时不安装这个程序，又该如何实现呢？

与上一节一样，主要还是通过模块的方法来实现这个需求，先建立一个名为 `nagioscli` 的模块，命令如下所示：

```
mkdir -p /etc/puppet/modules/nagiosins/{manifests,files,templates}
```

在 `/etc/puppet/modules/nagioscli/files` 目录下安装 Nagios 客户端名为 `nagioscli.sh` 的 Shell 程序，内容如下：

```

#!/bin/bash
useradd nagios
cd /usr/local/src
wget wget http://syslab.comsenz.com/downloads/linux/nagios-plugins-1.4.13.tar.gz
wget http://syslab.comsenz.com/downloads/linux/nrpe-2.12.tar.gz
tar zxvf nagios-plugins-1.4.13.tar.gz
cd nagios-plugins-1.4.13
./configure
make
make install
chown nagios:nagios /usr/local/nagios
chown -R nagios:nagios /usr/local/nagios/libexec
cd ../
tar zxvf nrpe-2.12.tar.gz
cd nrpe-2.12
./configure
make all
make install-plugin
make install-daemon

```

```
make install-daemon-config
sed -i 's@allowed_hosts=127.0.0.1@allowed_hosts=114.112.11.11@' /usr/local/nagios/etc/nrpe.cfg
/usr/local/nagios/bin/nrpe -c /usr/local/nagios/etc/nrpe.cfg -d
echo "/usr/local/nagios/bin/nrpe -c /usr/local/nagios/etc/nrpe.cfg -d" >> /etc/rc.local
```

node.pp 的文件内容如下:

```
node 'nginx.cn7788.com' {
  file
  {"/usr/local/src/nagiosins.sh":
    source => "puppet://server.cn7788.com/modules/nagiosins/nagiosins.sh",
    group => root,
    owner => root,
    mode => 755,
  }

  exec {
    "auto install nagios client":
    command => "sh /usr/local/src/nagiosins.sh",
    user => "root",
    path => ["/usr/bin", "/usr/sbin", "/bin", "/bin/sh"],
  }
}

node 'client.cn7788.com' {
  file
  {"/usr/local/src/nagiosins.sh":
    source => "puppet://server.cn7788.com/modules/nagiosins/nagiosins.sh",
    group => root,
    owner => root,
    mode => 755,
  }

  exec {
    "auto install nagios client":
    command => "sh /usr/local/src/nagiosins.sh",
    user => "root",
    path => ["/usr/bin", "/usr/sbin", "/bin", "/bin/sh"],
  }
}

node 'fabric.cn7788.com' {
}
```

若节点机器 client 和 fabric 机器后面什么都没有,则表示在此节点机器上没有进行任何操作,因为 client 和 fabric 节点机器也在此 Puppet 环境里,并配置成了自动连接,如此配置,是为了防止自动连接时 Puppet-Server 频繁报错。

这里以 client.cn7788.com 为例进行说明,在其主机上输入如下命令:

```
puppet agent --test --server server.cn7788.com
```


命令显示结果如下所示：

```
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Caching catalog for client.cn7788.com
Info: Applying configuration version '1446693418'
Notice: /Stage[main]/Main/Node[client.cn7788.com]/Exec[auto install nagios
      client]/returns: executed successfully
Notice: Finished catalog run in 165.27 seconds
```

可以看到，执行时间比较长，总共耗时 165.27 秒，因此要检查下 client.cn7788.com 的节点机器上是否开启了 nrpe 进程，输入如下命令：

```
ps aux | grep nrpe | grep -v grep
```

命令显示结果如下所示：

```
nagios    22331  0.0  0.1   5108   924 ?        Ss   22:35   0:00 /usr/local/
nagios/bin/nrpe -c /usr/local/nagios/etc/nrpe.cfg -d
```

下面检查 /etc/rc.local，看看此命令有没有添加进去，命令如下所示：

```
grep -v "^#" /etc/rc.local
```

命令执行结果显示如下：

```
touch /var/lock/subsys/local
/usr/local/nagios/bin/nrpe -c /usr/local/nagios/etc/nrpe.cfg -d
```

检查结果说明 Puppet-Master 的 nagioscli 模块是正常的，在 nginx.cn7788.com 上检测的结果类似，这里就不再贴出了。

5.5.6 如何快速同步 Puppet-Server 端的 www 目录文件

当有大规模的 Web 集群时，所有服务器上 /var/www/html 的数据要求迅速统一，那么该如何实现这一点呢？比如，server.cn7788.com 机器下的 /data/svn/resource 目录文件或子目录发生改变时，要求 nginx.cn7788.com、client.cn7788.com 及 fabric.cn7788.com 的 /var/www/html 对应目录也发生改变。

前面已经提到了，Puppet 对大文件和海量图片小文件进行分发的效果并不好，但其实可以用 rsync+Puppet 的方式来实现相应需求。这里要用到 Puppet kick 的知识点，即 Puppet-Server 端使用 puppet kick 命令强制 Puppet Agent 节点机器运行 puppet agent 命令，从而达到立即更新或同步文件的目的，当然也可以用 puppet rsync 模块，但笔者觉得使用这个太麻烦了，所以还是采用自己摸索出来的方法，具体步骤如下（这里以 client.cn7788.com 节点机器为例进行说明，其他 Puppet 客户端操作类似，就不再一一列举了）。

1) 在所有 Puppet-Client 机上配置 puppet.conf 文件，使其固定使用 8139 端口，然后在其 /etc/puppet/puppet.conf 文件下添加如下内容：

```
[puppet-client]
listen = true
server=server.cn7788.com
```

其中, `listen=true` 选项将使 puppet agent 监听 8139 端口; `server=server.cn7788.com` 选项也必须要配置, 经过测试可发现, 若无此选项, Puppet-Client 会连接不到 Puppet-Server 机器, 从而导致文件同步不过去。

2) 修改客户机端的 `/etc/puppet/auth.conf`, 允许 `server.cn7788.com` 的服务器端进行推送。

在 `auth.conf` 文件的最末行 `path /` 之后添加 `allow *`, 保证代码内容相同:

```
path /run
auth any
allow *
```

如果不进行此项操作的话, 会有如下报错:

```
Debug: /File[/var/lib/puppet/ssl/private]: Autorequiring File[/var/lib/puppet/ssl]
Debug: /File[/var/lib/puppet/ssl/certs/server.cn7788.com.pem]: Autorequiring
File[/var/lib/puppet/ssl/certs]
Debug: /File[/var/lib/puppet/ssl/private_keys]: Autorequiring File[/var/lib/puppet/ssl]
Debug: /File[/var/lib/puppet/lib]: Autorequiring File[/var/lib/puppet]
Debug: /File[/var/lib/puppet/ssl/private_keys/server.cn7788.com.pem]:
Autorequiring File[/var/lib/puppet/ssl/private_keys]
Debug: /File[/var/lib/puppet/ssl/certificate_requests]: Autorequiring File[/var/lib/
puppet/ssl]
Debug: /File[/var/lib/puppet/state]: Autorequiring File[/var/lib/puppet]
Debug: /File[/var/lib/puppet/ssl]: Autorequiring File[/var/lib/puppet]
Debug: /File[/var/lib/puppet/ssl/crl.pem]: Autorequiring File[/var/lib/puppet/ssl]
Debug: /File[/var/lib/puppet/facts.d]: Autorequiring File[/var/lib/puppet]
Debug: /File[/var/lib/puppet/ssl/certs]: Autorequiring File[/var/lib/puppet/ssl]
Debug: /File[/var/lib/puppet/ssl/public_keys]: Autorequiring File[/var/lib/puppet/ssl]
Debug: /File[/var/lib/puppet/preview]: Autorequiring File[/var/lib/puppet]
Debug: /File[/var/lib/puppet/ssl/certs/ca.pem]: Autorequiring File[/var/lib/puppet/
ssl/certs]
Debug: Finishing transaction 69854562006000
Debug: Creating new connection for https://client.cn7788.com:8139
Error: Host client.cn7788.com failed: Error 403 on SERVER: Forbidden request:
server.cn7788.com(192.168.1.205) access to /run/client.cn7788.com [save]
authenticated at :119
```

最后, 在 Puppet-Client 端重启 puppet 服务, 命令如下:

```
service puppet restart
```

3) 在 `server.cn7788.com` 机器的 `/etc` 目录下建立 `rsyncd.pass` 文件并分配内容。注意, 这个是推送到客户端的文件, 需要与 `/etc/rsyncd.password` 文件进行区分, `/etc/rsyncd.pass` 文件只需要指定同步用户的密码即可。 `/etc/rsyncd.password` 的文件内容如下:

```
test:test101
```

/etc/rsyncd.pass 的文件内容如下:

```
test101
```

4) 配置 Puppet-Server 端的 rsync 服务, /etc/rsyncd.conf 的文件内容如下:

```
uid = www
gid = www
user chroot= no
max connections =200
timeout = 600
pid file = /var/run/rsyncd.pid
lock file = /var/run/rsyncd.lock
log file = /var/log/rsyncd.log

[www]
path=/var/www/html/
ignore errors
read only = no
list = no
hosts allow = 192.168.1.0/255.255.255.0
auth users = test
secrets file = /etc/rsyncd.password
```

因为 Apache 服务的属主和属组是 www:www, 故而让 rsync 也以 www 用户运行, 这样可以保证通过 rsync 同步过去的文件的属性。这里采用 xinetd 管理的 rsync, 将其中的 disable 改为 no, 然后重启 xinetd 进程, 命令如下所示:

```
service xinetd restart
```

到了这一步其实还要仔细检查一下, 有时会因为存在配置文件的错误或文件权限分配的错误, 导致 rsync 进程并没有正确启动, 可用如下命令来检查:

```
lsof -i:873
```

命令显示结果如下, 这个结果表明 rsync 进程已经在监听 873 端口了, 服务已被正确启动了。

```
COMMAND  PID USER  FD   TYPE DEVICE SIZE/OFF NODE NAME
xinetd   7008 root    5u   IPv4  24249      0t0  TCP *:rsync (LISTEN)
```

5) 创建名为 wwwrsync 的模块, 命令如下:

```
mkdir -p /etc/puppet/modules/wwwrsync/{manifests,files,templates}
```

然后将 /etc/rsyncd.pass 复制至 /etc/puppet/modules/wwwrsync/files 目录下, 命令如下所示:

```
cp /etc/rsyncd.pass /etc/puppet/modules/wwwrsync/files/
```


6) 在 `/etc/puppet/modules/wwwrsync/manifests/init.pp` 里定义一个名为 `wwwrsync` 的类, `init.pp` 的文件内容如下:

```
class wwwrsync{
  package { httpd:
    ensure => present,
  }

  file {
    "/etc/rsyncd.pass":
    source => "puppet://server.cn7788.com/modules/wwwrsync/rsyncd.pass",
    owner => "root",
    group => "root",
    mode => "600",
  }

  exec {
    "auto rsync web directory":
    command => "rsync -vzrtopg --delete test@192.168.1.205::www /var/www/html
      --password-file=/etc/rsyncd.pass",
    user => "root",
    path => ["/usr/bin", "/usr/sbin", "/bin", "/bin/sh"],
  }
}
```

`init.pp` 文件中包含了 `wwwrsync` 的类, 此类又包含了 3 个资源, 第一个是名为 `httpd` 的 `package` 资源包, 如果没有安装此服务, Puppet 客户端会自行安装 `httpd` 服务, 保证在本机上自动生成 `/var/www/html` 目录; 第二个是 `file` 资源, 它会将 `/etc/puppet/modules/wwwrsync/files/rsyncd.pass` 文件推送到 Puppet-Client 端, 第三个是 `exec` 命令, 它会在 Puppet-Client 端执行 `rsync` 同步命令, 达到同步 `/var/www/html` 目录的目的, 所以 `rsync` 命令后面应该接 `rsync` 服务器地址, 即 `192.168.1.205`, 这点请大家注意不要弄混淆了。

 **注意** `wwwrsync` 模块中定义的 `wwwrsync` 类要跟 `wwwrsync` 模块同名, 不然 Puppet-Client 端在连接服务器端时会产生找不到 `wwwrsync` 类名的报错, 实验过程中如果遇到错误, 请注意多查看 Puppet 和系统日志。

7) 接着在 `/etc/puppet/manifests/site.pp` 中定义一个 `default` 的特殊节点, 这是一个默认节点, 它会将 `wwwrsync` 类中的内容应用到所有主机上面, 其内容如下所示:

```
node default {
  include wwwrsync
}
```

8) 在 `server.cn7788.com` 上面执行推送命令, 命令如下:

```
puppet kick -d --host `cat /etc/puppet/iplist.txt`
```




命令结果显示如下所示（这里只截取部分结果）：

```
Debug: /File[/var/lib/puppet/ssl/certs]/selrange: Found selrange default 's0' for
/var/lib/puppet/ssl/certs
Debug: /File[/var/lib/puppet/ssl/certs/ca.pem]: Autorequiring File[/var/lib/
puppet/ssl/certs]
Debug: /File[/var/lib/puppet/ssl/public_keys/server.cn7788.com.pem]: Autorequiring
File[/var/lib/puppet/ssl/public_keys]
Debug: /File[/var/lib/puppet/ssl/certs/server.cn7788.com.pem]: Autorequiring
File[/var/lib/puppet/ssl/certs]
Debug: /File[/var/lib/puppet/ssl/private_keys]: Autorequiring File[/var/lib/puppet/ssl]
Debug: /File[/var/lib/puppet/lib]: Autorequiring File[/var/lib/puppet]
Debug: /File[/var/lib/puppet/ssl/private_keys/server.cn7788.com.pem]: Autorequiring
File[/var/lib/puppet/ssl/private_keys]
Debug: /File[/var/lib/puppet/ssl/private]: Autorequiring File[/var/lib/puppet/ssl]
Debug: /File[/var/lib/puppet/ssl/crl.pem]: Autorequiring File[/var/lib/puppet/ssl]
Debug: /File[/var/lib/puppet/ssl/certificate_requests]: Autorequiring File[/var/lib/
puppet/ssl]
Debug: /File[/var/lib/puppet/state]: Autorequiring File[/var/lib/puppet]
Debug: /File[/var/lib/puppet/ssl]: Autorequiring File[/var/lib/puppet]
Debug: /File[/var/lib/puppet/ssl/certs]: Autorequiring File[/var/lib/puppet/ssl]
Debug: /File[/var/lib/puppet/facts.d]: Autorequiring File[/var/lib/puppet]
Debug: /File[/var/lib/puppet/ssl/public_keys]: Autorequiring File[/var/lib/puppet/ssl]
Debug: /File[/var/lib/puppet/preview]: Autorequiring File[/var/lib/puppet]
Debug: Finishing transaction 69995814564700
Debug: Creating new connection for https://client.cn7788.com:8139
Getting status
status is success
client.cn7788.com finished with exit code 0
```

/etc/puppet/iplist.txt 的文件内容如下：

```
client.cn7788.com
nginx.cn7788.com
fabric.cn7788.com
```

如果观察名为 client.cn7788.com 的节点机器，会发现它的 /var/www/html 文件也立即跟 server.cn7788.com 的 /var/www/html 目录同步了，从而实现了此需求，用 tail 命令观察 client.cn7788.com 机器的 messages 日志，结果如下所示：

```
Nov 5 03:26:46 client puppet-agent[27782]: (/Stage[main]/wwwrsync/Exec[auto
rsync web directory]/returns) executed successfully
Nov 5 03:26:47 client puppet-agent[27782]: Finished catalog run in 0.74 seconds
Nov 5 03:41:50 client puppet-agent[28422]: (/Stage[main]/wwwrsync/Exec[auto
rsync web directory]/returns) executed successfully
Nov 5 03:41:51 client puppet-agent[28422]: Finished catalog run in 1.32 seconds
Nov 5 04:11:50 client puppet-agent[28690]: (/Stage[main]/wwwrsync/Exec[auto
rsync web directory]/returns) executed successfully
Nov 5 04:11:50 client puppet-agent[28690]: (/Stage[main]/wwwrsync/File[/etc/
rsyncd.pass]/content) content changed '{md5}d8e8fca2dc0f896fd7cb4cb0031ba249'
to '{md5}93412aea2e70977a362530b0dba2498a'
```



```

Nov  5 04:11:52 client puppet-agent[28690]: Finished catalog run in 1.97 seconds
Nov  5 04:14:17 client puppet-agent[27782]: triggered run
Nov  5 04:14:25 client puppet-agent[27782]: (/Stage[main]/wwwrsync/Exec[auto
rsync web directory]/returns) executed successfully
Nov  5 04:14:26 client puppet-agent[27782]: Finished catalog run in 1.69 seconds

```

5.5.7 如何利用 ERB 模板来自动配置 Apache 虚拟主机

线上环境中有不少 Apache 主机需要增加基于域名的虚拟主机，特别是有的虚拟主机达到几十台之多，那么如何才能方便快捷地部署 httpd.conf 和虚拟主机配置文件呢？

这里就需要用到 Puppet 的 ERB 模板功能了，模板文件就是在模块下面的 templates 目录中以“.erb”结尾的文件，Puppet 模板主要用于文件，例如各种服务的配置文件，如果有着相同的服务，又需要进行不同的配置，就可以考虑使用该模板文件了，像 Nginx 和 Apache 的虚拟主机配置就可以考虑采用 ERB 模板的方案，这里以 Apache 为例来说明下，其模块目录树结构如下：

```

/etc/puppet/
├── auth.conf
├── environments
│   └── example_env
│       ├── manifests
│       └── modules
│           └── README.environment
├── fileserver.conf
├── manifests
│   ├── nodes
│   │   ├── client.pp
│   │   └── nginx.pp
│   └── site.pp
├── modules
│   └── apache
│       ├── files
│       ├── manifests
│       │   └── init.pp
│       └── templates
│           └── httpd.conf.erb
└── puppet.conf

```

11 directories, 9 files

Apache 模块的具体配置步骤如下。

首先建立名为 apache 的模块，命令如下：

```
mkdir -p /etc/puppet/modules/apache/{files,manifests,templates}
```

这时，/etc/puppet/modules/apache/manifests/init.pp 文件内容如下：

```

class apache{
    package{"httpd":
        ensure =>present,
    }

    service{"httpd":
        ensure      =>running,
        require     =>Package["httpd"],
    }
}

```

```

define apache::vhost ( $sitedomain, $rootdir,$port ) {
    file { ["/etc/httpd/conf.d/httpd_vhost_${sitedomain}.conf":
        #path      => '/etc/httpd/conf/httpd_vhost.conf',
        content => template("apache/httpd.conf.erb"),
        require => Package["httpd"],
    ]
}

```

这里用到了 Puppet 中的 define（定义）概念，定义和类也属于资源容器，不过它的特点是能够在一台主机上被赋值多次，此外它还能接受参数。类在 Puppet 中是单例的，它们能够在一台主机节点机器上被包含多次，但是只会被求值一次；而定义因为能够接受参数，所以可以被声明多次，并且每一个声明都会被求值。

/etc/puppet/modules/apache/templates 中的 httpd.conf.erb 模板文件内容如下：

```

<VirtualHost *:<%= port %>>
ServerName <%= sitedomain %>
DocumentRoot /var/www/html/<%= rootdir %>
<Directory <%= rootdir %>>
Options Indexes FollowSymLinks
AllowOverride None
Order allow,deny
Allow from all
</Directory>
ErrorLog logs/<%= sitedomain %>_error.log
CustomLog logs/<%= sitedomain %>_access.log common
</VirtualHost>

```

httpd.conf.erb 里提前定义了两个变量 \$sitedomain 和 \$port，Puppet 中使用如下格式：“<%= 变量名 %>”，可以用如下命令来检测模板是否存在语法问题。

```
erb -x -T '-' -P /etc/puppet/modules/apache/templates/httpd.conf.erb | ruby -c
```

结果显示如下，表示语法不存在任何问题：

```
Syntax OK
```

Puppet-Server 机器的 /etc/puppet/manifests/site.pp 内容如下所示：

```
import 'nodes/*.pp'
```

其 nodes 目录里面有两个文件，一个为 client.pp，另一个为 nginx.pp。Puppet-Server 机

器的 /etc/puppet/manifests/nodes/nginx.pp 文件内容如下:

```
node 'nginx.cn7788.com' {
  include apache
  apache::vhost {'webmaster.cn7788.com':
    sitedomain => "webmaster.cn7788.com",
    rootdir => webmaster,
    port => 80,
  }

  apache::vhost {'webtest.cn7788.com':
    sitedomain => "webtest.cn7788.com",
    rootdir => webtest,
    port => 80,
  }
}
```

```
apache::vhost {'webrsync.cn7788.com':
  sitedomain => "webrsync.cn7788.com",
  rootdir => webrsync,
  port => 80,
}
```

另一台节点机器 client.cn7788.com 的 client.pp 配置文件如下所示:

```
node 'client.cn7788.com' {
  include apache
  apache::vhost {'clientmaster.cn7788.com':
    sitedomain => "webmaster.cn7788.com",
    rootdir => webmaster,
    port => 80,
  }

  apache::vhost {'clienttest.cn7788.com':
    sitedomain => "webtest.cn7788.com",
    rootdir => webtest,
    port => 80,
  }
}
```

在 nginx.cn7788.com 的机器上输入如下命令进行验证:

```
puppet agent --test --server server.cn7788.com
```

显示结果如下所示:

```
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Caching catalog for nginx.cn7788.com
```

```

Info: Applying configuration version '1446792027'
Notice: /Stage[main]/Apache/Package[httpd]/ensure: created
Notice: /Stage[main]/Main/Node[nginx.cn7788.com]/Apache::Vhost[webtest.cn7788.com]/File[/etc/httpd/conf.d/httpd_vhost_webtest.cn7788.com.conf]/ensure: defined content as '{md5}d5befc97a115a5069b6f8fd7e904b919'
Notice: /Stage[main]/Main/Node[nginx.cn7788.com]/Apache::Vhost[webmaster.cn7788.com]/File[/etc/httpd/conf.d/httpd_vhost_webmaster.cn7788.com.conf]/ensure: defined content as '{md5}07017828a5d085223c6635afff1d0f69'
Notice: /Stage[main]/Apache/Service[httpd]/ensure: ensure changed 'stopped' to 'running'
Info: /Stage[main]/Apache/Service[httpd]: Unscheduling refresh on Service[httpd]
Notice: /Stage[main]/Main/Node[nginx.cn7788.com]/Apache::Vhost[webrsync.cn7788.com]/File[/etc/httpd/conf.d/httpd_vhost_webrsync.cn7788.com.conf]/ensure: defined content as '{md5}5fa6f92aa7ebb670a08e6e0968df9be6'
Notice: Finished catalog run in 87.65 seconds

```

该结果表示配置是成功的，整个过程耗时 87.65 秒。



注意 很多资料和文档都是复制 `/etc/httpd/conf/httpd.conf` 文件来作为 `httpd.conf.erb` 模板的，个人觉得这种做法还是欠缺考虑，一般来说，每台 Apache 主机上面至少有一个基于域名的虚拟主机，有时更多，十几个也很常见，所以才需要用独立的虚拟主机文件来管理虚拟主机并自动载入（请注意配置文件 `httpd.conf` 中存在着这么一行内容“`include conf.d/*.conf`”，这个指令的意思就是将 `conf.d` 目录下所有以 `.conf` 结尾的文件都引进来），这也是利用 ERB 模板文件将虚拟主机的文件定义路径放在 `/etc/httpd/conf.d` 目录下的原因。

5.5.8 如何利用 ERB 模板来自动配置 Nginx 虚拟主机

如果 Web 集群环境已经上线，那么应该如何方便快速地部署 Nginx 及其虚拟主机呢？要想实现这个需求可以参考 5.5.6 节的内容，这里的 Nginx 建议采用第三方 yum 源来安装。如果是用 Nginx 的官方源来安装，可以添加如下内容到 `/etc/yum.repos.d/nginx.repo` 文件：

```

[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/centos/$releasever/$basearch/
gpgcheck=0
enabled=1

```

第二种方式就是通过 `createrepo` 命令建立自己的 yum 源，这种方式更加灵活，可以先在 Nginx 官网上下载适合自己的 rpm 源码包，然后通过执行 `rpmbuild` 命令使其成为 rpm 包，并添加进自己的 yum 源，在自动化运维要求严格的定制环境中，绝大多数运维人员都会选择这种方法。通过此种方式安装 Nginx 以后会发现，确实比源码安装方便多了，比如，可以自动分配运行 Nginx 的用户 `nginx`。另外，Nginx 的日志存放会自动保存在 `/var/log/nginx` 下，其工作目录为 `/etc/nginx`，这一点跟源码编译安装的 Nginx 区别比较大，请大家

注意区分。

Puppet-Server 机器的 /etc/puppet 文件结构如下：

```

├── auth.conf
├── environments
│   └── example_env
│       ├── manifests
│       ├── modules
│       └── README.environment
├── fileserver.conf
├── manifests
│   ├── nodes
│   │   ├── client.cn7788.com.pp
│   │   └── nginx.cn7788.com.pp
│   └── site.pp
├── modules
│   └── nginx
│       ├── files
│       ├── manifests
│       │   └── init.pp
│       └── templates
│           ├── nginx.conf.erb
│           └── nginx_vhost.conf.erb
└── puppet.conf

```

首先建立 Nginx 模块，命令如下：

```
mkdir -p /etc/puppet/modules/nginx/{files,manifests,templates}
```

Nginx 模块的配置文件挺多，这里将详细说明一下。

site.pp 的文件内容如下：

```
import "nodes/*.pp"
```

client.cn7788.com.pp 的文件内容如下：

```

node 'client.cn7788.com' {
  include nginx
  nginx::vhost {'client.cn7788.com':
    sitedomain => "client.cn7788.com",
    rootdir => "client",
  }
}

```

nginx.cn7788.com.pp 的文件内容如下：

```

node 'nginx.cn7788.com' {
  include nginx
  nginx::vhost {'nginx.cn7788.com':
    sitedomain => "nginx.cn7788.com",
    rootdir => "nginx",
  }
}

```




```
}
```

`/etc/puppet/modules/nginx/manifests/init.pp` 的文件内容如下:

```
class nginx{
    package{"nginx":
        ensure =>present,
    }

    service{"nginx":
        ensure      =>running,
        require     =>Package["nginx"],
    }

    file{"nginx.conf":
        ensure => present,
        mode => 644,
        owner => root,
        group => root,
        path => "/etc/nginx/nginx.conf",
        content=> template("nginx/nginx.conf.erb"),
        require=> Package["nginx"],
    }

    define nginx::vhost($sitedomain,$rootdir) {
        file{ "/etc/nginx/conf.d/${sitedomain}.conf":
            content => template("nginx/nginx_vhost.conf.erb"),
            require => Package["nginx"],
        }
    }
}
```

`/etc/puppet/modules/nginx/templates/nginx.conf.erb` 的文件内容如下:

```
user nginx;
worker_processes 8;
error_log /var/log/nginx/error.log warn;
pid /var/run/nginx.pid;
events {
    use epoll;
    worker_connections 51200;
}

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
        '$status $body_bytes_sent "$http_referer" '
        '"$http_user_agent" "$http_x_forwarded_for"';

    access_log /var/log/nginx/access.log main;
    sendfile on;
    #tcp_nopush on;
    keepalive_timeout 65;
    #gzip on;
    include /etc/nginx/conf.d/*.conf;
}
```

然后检查下此 ERB 模板文件的语法，命令如下：

```
erb -x -T '-' -P /etc/puppet/modules/apache/templates/nginx.conf.erb | ruby -c
```

如果没有任何显示，就说明文件在语法上是不存在任何问题的。

/etc/puppet/modules/nginx/templates/nginx_vhost.conf.erb 的文件内容如下：

```
server {
  listen      80;
  server_name <%= sitedomain %>;
  access_log /var/log/nginx/<%= sitedomain %>.access.log;
  location / {
    root /var/www/<%= rootdir %>;
    index index.php index.html index.htm;
  }
}
```

最后可以在节点名为 client.cn7788.com 和 nginx.cn7788.com 的机器上验证效果，命令如下：

```
puppet agent --test --server server.cn7788.com
```

这里以 nginx.cn7788.com 节点机器为例说明，此命令执行结果显示如下：

```
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Caching catalog for nginx.cn7788.com
Info: Applying configuration version '1446798263'
Notice: /Stage[main]/Nginx/Package[nginx]/ensure: created
Notice: /Stage[main]/Nginx/Service[nginx]/ensure: ensure changed 'stopped' to 'running'
Info: /Stage[main]/Nginx/Service[nginx]: Unscheduling refresh on Service[nginx]
Notice: /Stage[main]/Main/Node[nginx.cn7788.com]/Nginx::Vhost[nginx.cn7788.com]/
  File[/etc/nginx/conf.d/nginx.cn7788.com.conf]/ensure: defined content as '{md
  5}5f08d10788e3c82b41336a40edc5350f'
Notice: /Stage[main]/Nginx/File[nginx.conf]/content:
--- /etc/nginx/nginx.conf 2015-04-21 15:34:33.0000000000 +0000
+++ /tmp/puppet-file20151106-5957-1f964a8-0 2015-11-06 08:27:14.267072983 +0000
@@ -1,32 +1,22 @@
-
- user nginx;
- worker_processes 1;
-
+ worker_processes 8;
+ error_log /var/log/nginx/error.log warn;
+ pid /var/run/nginx.pid;
-
- events {
-   worker_connections 1024;
+   use epoll;
+   worker_connections 51200;
```

```

}
-
http {
    include                /etc/nginx/mime.types;
    default_type            application/octet-stream;

    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
                    '$status $body_bytes_sent "$http_referer" '
                    '"$http_user_agent" "$http_x_forwarded_for"';

    access_log /var/log/nginx/access.log main;

    sendfile                on;
    #tcp_nopush              on;

    keepalive_timeout      65;

    #gzip                    on;

    include /etc/nginx/conf.d/*.conf;
}
+

```

```

Info: Computing checksum on file /etc/nginx/nginx.conf
Info: FileBucket got a duplicate file {md5}f7984934bd6cab883elf33d5129834bb
Info: /Stage[main]/Nginx/File[nginx.conf]: Filebucketed /etc/nginx/nginx.conf to
puppet with sum f7984934bd6cab883elf33d5129834bb
Notice: /Stage[main]/Nginx/File[nginx.conf]/content: content changed '{md5}f7984
934bd6cab883elf33d5129834bb' to '{md5}34e85800459aaf9b40ebfbdafa33614c0'
Notice: Finished catalog run in 42.19 seconds

```

在 nginx.cn7788.com 的机器上检查生成的 Nginx 相关配置文件，发现都已经顺利生成了，说明 Nginx 模板配置是成功的。


5.6 Puppet 的负载均衡方式

此外，关于 Puppet 的负载均衡方式，这里也要说明一下。

随着公司应用需求的增加，服务器数量也随之增加，当服务器数量不断增加时，我们会发现一台 Puppet Server 的压力变大，解析缓慢，而且时不时还会出现“timeout”之类的报错，那么有什么解决方法吗？在 Puppet 官网上寻找解决方案，会发现 Puppet Server 可以配置多端口，结合轻量级的 Nginx 代理，这样 Puppet 的承受能力至少可以提升数倍以上，相当于在很大程度上优化了 Puppet 的并发处理能力。

其实 Nginx+Mongrel 模式的原理很简单，即：通过 Nginx 负载均衡 Puppet Server 的进程，由 Nginx 向所有的 Puppet Agent 提供认证服务，除此之外的其他 Puppet Server 功

能的实现，将由 Nginx 转向 Puppet Server 中的另一个进程处理完成，这样就极大地提升了 Puppet 的并发处理能力。

 **说明** Puppet 3.0 及 3.0 以上的版本不再支持 Mongrel 模式，改用 Nginx+Passenger 模式，这一点也请注意区别。

5.7 用 GitHub 来管理 Puppet 配置文件

随着 Puppet 节点机器的增多，其模块配置文件也越来越多，越来越不方便管理了，这个时候可以利用 SVN 或 Git 这些代码版本控制软件来管理 Puppet 的相关配置文件。

Git 与 SVN 相比而言，优势还是很明显的，具体如下：

- ☐ Git 是分布式的，而 SVN 是集中式的。
- ☐ Git 可以在无网络的环境下提交，可以进行离线代码提交，因此称得上是完全的分布式处理。Git 的所有操作都不需要在线进行，这就意味着 Git 的速度要比 SVN 等工具快得多，因为 SVN 等工具需要在线时才能操作，如果网络环境不好，提交代码会变得非常缓慢。
- ☐ Git 的分支功能要比 SVN 强大得多，事实上，分支模型是 Git 最显著的特点。
- ☐ 解决冲突方面 Git 也比 SVN 更方便。
- ☐ GitHub 现在越来越流行了，像笔者所在公司采用的就是付费的 GitHub 私有库来进行代码托管的，线上代码的管理工作感觉相当稳定和方便。

综上所述，这里也推荐大家采用 GitHub 的方式来管理 Puppet 配置文件代码，Git 命令在 Mac 和 Ubuntu 系统下面都已经自带安装了，如果大家的办公机器是 Win8 或 Win10 系列，推荐用 msysGit，msysGit 是 Git 控制系统在 Windows 下的版本，下载地址为 <https://git-for-windows.github.io/>，为了节约篇幅，具体安装过程略过。

GitHub 网址为 <https://github.com>，大家可以自行注册，然后建立自己的版本库，笔者这里采用的是 <http://github.com/yuhongchun/avaliablity>。

下面来具体看下 msysGit 的使用步骤：

1) 在电脑的特定位置上（笔者这里是 Windows 8.1 x86_64 的桌面）点击鼠标右键选择“Git Bash Shell”，输入 pwd 查看当前位置，命令显示结果如下所示：

```
/c/Users/洪春/Desktop
```

2) 创建 SSH Keys 文件，输入如下所示命令：

```
ssh-keygen -t rsa -C "yuhongchun027@gmail.com"
```

3) 输入后连续按 3 个回车，就可以在默认的文件夹下生成 keys 文件，命令显示结果如下：

```
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/洪春/.ssh/id_rsa):
Created directory '/c/Users/洪春/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/洪春/.ssh/id_rsa.
Your public key has been saved in /c/Users/洪春/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:YR/fRVUIUZHJPNl89RB36bTyJ+LqXIQYxfpwh05v0U yuhongchun027@gmail.com
The key's randormart image is:
+---[RSA 2048]-----+
|           .oOBoo*|
|           .B. +Eo|
|           + +.+. B|
|           . O *.+oBo|
|           S * . O+.|
|           . . oo|
|           . . .|
|           . . o .|
|           o..o .|
+---[SHA256]-----+
```

4) 用编辑器打开 id_rsa.pub 文件，这里采用的是 Sublime Text 3，将其内容全部复制。然后打开自己的 GitHub 主页，选择“SSH Keys”菜单，点击“Add SSH key”选项，在最下面的空白框里复制刚才选择的公钥文件内容，还可以取一个自己定义的名字，如图 5-3 所示。



图 5-3 GitHub 添加 SSH Key 图示

5) 回到 Git Bash 当中，输入如下命令进行测试：


```
ssh -T git@github.com
```

命令显示结果如下：

```
Hi yuhongchun! You've successfully authenticated, but GitHub does not provide shell access.
```

结果表示一切正常，接下来就可以通过 `git clone` 命令加载自己的版本库了，命令如下所示：

```
git clone git@github.com:yuhongchun/avaliablity.git
```

注意这里的格式，如果写成 `git://github.com/yuhongchun/avaliablity.git`，那么客户端的 URL 请求是可读模式，写成 `git@github.com:yuhongchun/avaliablity.git` 这样的格式才是可写的模式。

当然了，也可以选择 HTTPS 协议，但 Git 协议是 Git 使用的网络传输协议里最快的，这也是我没有选择 HTTPS 协议的原因，命令执行成功以后，就本地的办公机器上克隆一个本地版本库。

6) 向 GitHub 提交自己的名字和邮箱。

在此之前还需要设置 `username` 和 `email`，因为 GitHub 每次提交都会记录它们，下面笔者还是以自己的名字和 E-mail 邮箱地址来举例说明，命令如下：

```
git config --global user.name "your name"
git config --global user.email "your_email@youremail.com"
```

7) 向自己的 GitHub 提交代码，以 `init.pp` 文件为例，先将 `init.pp` 复制粘贴到本地的 `security` 目录下面，执行下面的步骤：

```
git add init.pp #添加init.pp至本地版本库的暂时区
git commit -m "the apache module" #向本地版本库提交init.pp
git push #向自己的GitHub版本库提交代码
```

正确提交 `init.pp` 以后，就可以在其 GitHub 上面看到所提交的代码，也就可以像管理我们的代码一样管理其 Puppet 配置文件了，如图 5-4 所示。

另外，补充一个小知识点，要是想把 Apache 整个目录 `git push` 至 GitHub 上面去，却发现不能上传下面的空目录 `files`，这里什么原因呢？

具体原因如下：Git 和 SVN 不同，它仅跟踪文件的变动，而不跟踪目录。所以，一个空目录如果里面没有文件，即便 Git 加载了这个目录，也是没有任何效果的，版本库是不会做任何记录的。只跟踪文件变化，不跟踪目录，Git 这么设计是有原因的，但这也会带来一些小麻烦。有时候，确实需要在代码仓库中保留某个空目录。比如，若要提交 Apache 的所有子目录就会出现问题，但我们可以采用一个变通的解决办法，那就是在空目录 `files` 下存一个 `.gitignore` 文件（`.gitignore` 文件的作用是列出不希望 Git 跟踪的文件和文件夹）。做好这些以后，就可以顺利执行 `git add` 和 `git push` 命令了，步骤如下：

My_Python	upload	19 hours ago
.gitignore	upload	19 hours ago
check_cpu_counter.sh	upload	19 hours ago
check_cpu_utilization.sh	upload	19 hours ago
check_ip_connects.sh	upload	19 hours ago
check_redis.py	upload	19 hours ago
check_sync_redis.sh	upload	19 hours ago
definition.py	upload	19 hours ago
echo_limit.sh	upload	19 hours ago
filter-firewall.sh	upload	19 hours ago
ftp.py	upload	19 hours ago
init.pp	thd apache module	2 minutes ago
lambda.py	upload	19 hours ago
nginx-ha.sh	upload	19 hours ago
ping.py	upload	19 hours ago
rsync_host_file.py	upload	19 hours ago
rsync_nagios_ipcon.py	upload	19 hours ago
rsync_nagios_nrpe.py	upload	19 hours ago

图 5-4 GitHub 网站版本库文件明细图

```
git add apache
git commit -m "the apache module"
git push
```

顺利的话，在自己的 GitHub 版本库上就能看到 Apache 目录的所有子目录及其文件了。

一般来说，公司的项目或网站会分成三种环境：开发环境、测试环境和线上环境，将其放在哪个环境的 Git 中，这个就要视公司的环境而定了，就笔者的公司而言，线上环境的变动是最大的，所以主要是利用线上环境的 Git 来对 Puppet 配置文件进行管理，在每一次提交其 Puppet 改动后的配置文件时，笔者都会记录其详细的 git log 日志，这样回退 Puppet 配置文件的旧版本时就有据可查了。

5.8 小结

分布式自动化部署管理工具 Puppet 这个软件越来越成熟和强大了，它有着很好的发展前景，由于业务环境的关系，这里只简单介绍了自动化部署管理工具 Puppet 的安装、部署及平时工作的常见用法，像 Puppet 的控制台产品 Dashboard 和 Foreman 都没有涉及，有兴趣的朋友可以结合实际工作尝试研究 Puppet 更高级的用法。

Linux 防火墙及系统安全篇

系统安全可以说是系统运维的最基本要求了，如果连这一点都保证不了，其他运维工作都是空谈。这一章首先会向大家介绍 Linux 下的防火墙 iptables 的详细使用方法，它对系统安全的作用，以及它在普通服务器和 AWS EC2 云主机上的应用，然后会介绍 Linux 服务器下的安全防护技术。初学 Linux 防火墙的读者朋友可能会觉得 iptables 语法复杂，又是在纯字符下操作的（AWS EC2 云主机的安全组件操作是图形化操作，较容易上手），不易学习，其实只要掌握正确的学习方法，严格按照 iptables 的语法规则来执行，循序渐进，上手也是件很容易的事情。学习 iptables 跟学习英语一样，都是有语法和规律可言的，建议大家参考笔者所提供的 iptables 学习脚本和 iptables 线上脚本来学习，在了解 iptables 的语法规则后，相信很快就可以掌握 iptables 的用法了。

6.1 基础网络知识

这一节将向大家介绍关于网络的几个知识点，理解这些基础的知识点，对理解以后的 Linux 防火墙 iptables 的工作流程会很有帮助。

6.1.1 OSI 网络参考模型

OSI（Open System Interconnection）模型表示一种层次型的网络架构。该模型共有 7 层，每一层都有其独特的功能（如图 6-1 所示）。

各层的作用具体说明如下。



图 6-1 OSI 模型的 7 层参考模型

- ❑ 物理层：主要定义物理设备标准，如网线的接口类型、光纤的接口类型、各种传输介质的传输速率等。它的主要作用是传输比特流（即由 1、0 转化为强弱电流来进行传输，到达目的地后再转化为 1、0，也就是我们常说的模数转换与数模转换）。这一层的数据称为比特，网卡工作在此层。物理层一般较少关心网络入侵分析，更关注于保证设备的电缆安全。
- ❑ 数据链路层：主要是将从物理层接收的数据进行 MAC 地址（网卡的地址）的封装与解封装。这一层的数据常被称为帧。在这一层工作的设备是交换机，数据通过交换机来传输（三层交换机不在此层工作）。
- ❑ 网络层：主要用于将从下层接收到的数据进行 IP 地址（例如 192.168.0.1/24）的封装与解封装。在这一层工作的设备是路由器，这一层的数据常被称为数据包。
- ❑ 传输层：本层定义了一些传输数据的协议和端口号（如 www 端口 80 等），比如：TCP（传输控制协议，传输效率低，可靠性强，用于传输对可靠性要求高且数据量大的数据）和 UDP（用户数据报协议，与 TCP 的特性恰恰相反，传输的是对可靠性要求不高且数据量小的数据，如 QQ 聊天数据）等。传输层主要是将从下层接收的数据进行分段传输，到达目的地后再重组。我们常把这一层的数据称为段。
- ❑ 会话层：通过传输层（端口号：传输端口与接收端口）建立数据传输的通路。主要是在系统之间发起会话或接受会话请求（设备之间可通过 IP，也可通过 MAC 或主机名来相互认识）。
- ❑ 表示层：主要是对接收的数据进行解释、加密与解密、压缩与解压缩等操作（也就是把计算机能够识别的东西转换成人能够识别的东西，比如图片、声音等）。
- ❑ 应用层：主要是一些终端的应用，比如说 FTP（文件下载）、Web（IE 浏览）、QQ 之类的应用（也可以把它理解成我们在电脑屏幕上所看到的东西，也就是终端应用），也可以理解为应用层是负责向用户或应用程序显示数据的。

6.1.2 TCP/IP 三次握手的过程详解

1. 建立连接协议（三次握手）

- 1) 客户端发送一个带 SYN 标志的 TCP 报文到服务器。这是三次握手过程中的报文 1。
- 2) 服务器端回应客户端，这是三次握手中的第 2 个报文。这个报文同时带有 ACK 标志和 SYN 标志，它表示对刚才客户端 SYN 报文的回应，同时又将标志 SYN 回传给客户端，询问客户端是否准备好进行数据通信。
- 3) 客户端必须再次回应服务器端一个 ACK 报文，这是报文 3。



说明 报文（message）是网络中交换与传输的数据单元，其中包含了将要发送的完整的数据信息，其长短很不一致。报文又可分为自由报文和数字报文两种。它也是网络传输的单位，会不断地通过封装成分组、包、帧来进行传输，封装的方式就是添加一些信息段，所添加的信息段就是报文头。

2. 连接终止协议（四次挥手）

由于 TCP 连接是全双工的，因此每个方向都必须单独进行关闭。原则上是当一方完成它的数据发送任务时就发送一个 FIN 来终止这个方向的连接。收到一个 FIN 只意味着这一方向上没有数据流动了，一个 TCP 连接在收到一个 FIN 后仍能发送数据。首先关闭的一方将执行主动关闭，而另一方则执行被动关闭。具体步骤如下：

1) TCP 客户端发送一个 FIN，用来关闭客户到服务器的数据传送（报文 4）。

2) 服务器收到这个 FIN，它发回一个 ACK，确认序号为收到的序号加 1（报文 5）。和 SYN 一样，一个 FIN 将占用一个序号。

3) 服务器关闭客户端的连接，发送一个 FIN 给客户端（报文 6）。

4) 客户端发回 ACK 报文确认，并将确认序号设置为收到的序号加 1（报文 7）。

在 TCP 三次握手、四次挥手的过程中存在多种 TCP 连接状态，分别如下。

❑ CLOSED：这个没什么好说的，表示初始状态。

❑ LISTEN：这也是非常容易理解的一个状态，表示服务器端的某个 Socket 正处于监听状态，可以接收连接了。

❑ SYN_RCVD：这个状态表示接收到了 SYN 报文，在正常情况下，这个状态是服务器端的 Socket 在建立 TCP 连接时的三次握手会话过程中的一个中间状态，很短暂，用 netstat 命令基本上是很难看到这种状态的，除非特意编写一个客户端测试程序，故意将 TCP 三次握手过程中的最后一个 ACK 报文不予发送。在正常情况下，当收到客户端的 ACK 报文时，它会进入到 ESTABLISHED 状态。

❑ SYN_SENT：这个状态与 SYN_RCVD 遥相呼应，当客户端 Socket 执行 CONNECT 连接时，它首先会发送 SYN 报文，随即进入 SYN_SENT 状态，并等待服务端发送三次握手中的第 2 个报文。SYN_SENT 状态表示客户端已发送了 SYN 报文。

❑ ESTABLISHED：这个很容易理解，表示连接已经建立了。

❑ FIN_WAIT_1：这个状态要好好解释一下，其实 FIN_WAIT_1 和 FIN_WAIT_2 状态的真正含义都表示正在等待对方的 FIN 报文。而这两种状态的区别是：FIN_WAIT_1 状态实际上是当 Socket 在 ESTABLISHED 状态时，它想主动关闭连接，于是向对方发送了 FIN 报文，然后该 Socket 就进入到 FIN_WAIT_1 状态了。而在对方回应 ACK 报文后，则进入 FIN_WAIT_2 状态，当然在实际的正常情况下，无论对方处于何种情况，都应该马上回应 ACK 报文，所以 FIN_WAIT_1 状态一般是很难看到的，而 FIN_WAIT_2 状态还是常常可以用 netstat 看到的。

❑ FIN_WAIT_2：上面已经详细解释了这种状态，实际上 FIN_WAIT_2 状态下的 Socket，表示半连接，即有一方要求关闭连接，同时还会告诉对方，我暂时还有点数据需要传给你，稍后将关闭连接。

❑ TIME_WAIT：表示收到了对方的 FIN 报文，并发送出了 ACK 报文，就等 2MSL 后回到 CLOSED 可用状态。如果 FIN_WAIT_1 状态下，收到了对方同时带有 FIN 标志和

ACK 标志的报文，就可以直接进入 `TIME_WAIT` 状态了，而无须经过 `FIN_WAIT_2` 状态。

❑ `CLOSING`：这种状态比较特殊，在实际应用中应该很少见，属于一种比较罕见的例外状态。正常情况下，当你发送 `FIN` 报文时，按理来说是应该先收到（或同时收到）对方的 `ACK` 报文，再收到对方的 `FIN` 报文。但是 `CLOSING` 状态表示你发送 `FIN` 报文后，并没有收到对方的 `ACK` 报文，而是收到了对方的 `FIN` 报文。在什么情况下会出现此状况呢？其实细想一下，也不难得出结论：如果双方几乎在同时关闭一个 `Socket` 的话，那么就会出现双方同时发送 `FIN` 报文的情况，也就会出现 `CLOSING` 状态，表示双方都正在关闭 `Socket` 连接。

❑ `CLOSE_WAIT`：这种状态的含义其实是表示正在等待关闭。怎么理解呢？当对方关闭一个 `Socket` 后发送 `FIN` 报文给自己，系统会毫无疑问地回应一个 `ACK` 报文给对方，此时就会进入到 `CLOSE_WAIT` 状态。实际上接下来你真正需要考虑的事情是，查看你是否还有数据要发送给对方，如果没有的话，那么你就可以关闭这个 `Socket`，发送 `FIN` 报文给对方，即关闭连接了。所以在 `CLOSE_WAIT` 的状态下，需要完成的事情是等待，然后关闭连接。

❑ `LAST_ACK`：这个状态还是比较好理解的，它是被动关闭的一方在发送 `FIN` 报文后，最后等待对方的 `ACK` 报文。当收到 `ACK` 报文时，就表示可以进入 `CLOSED` 可用状态了。

现在，大家来思考一个问题：为什么 `TIME_WAIT` 状态还需要等 `2MSL`（报文最大生存时间）后才能返回到 `CLOSED` 状态呢？

答案是虽然双方都同意关闭连接了，而且握手的 4 个报文也都协调好并发送完毕，按理可以直接回到 `CLOSED` 状态（就好比从 `SYN_SEND` 状态到 `ESTABLISH` 状态一样），但是因为我们必须要假设网络是不可靠的，你无法保证最后发送的 `ACK` 报文一定会被对方收到，比如对方正处于 `LAST_ACK` 状态下的 `Socket` 可能会因为超时未收到 `ACK` 报文，这时就需要重发 `FIN` 报文，所以这个 `TIME_WAIT` 状态的作用就是用来重发可能丢失了的 `ACK` 报文的。

希望下面的流程图能够帮助大家理解这个过程，流程图如图 6-2 所示。

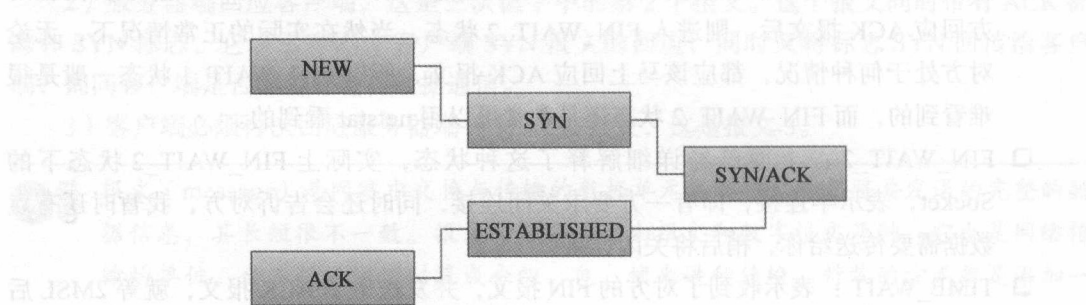


图 6-2 TCP/IP 三次握手流程图（顺序为 `NEW` → `SYN/ACK` → `ACK`）

6.1.3 Socket 应用及其他基础网络知识

随着 TCP/IP 协议的使用, Socket (套接字) 也被越来越多地使用到网络应用程序的构建中。实际上, Socket 编程已经成为了网络中传送和接收数据的首选方法。Socket 相当于应用程序访问下层网络服务的接口, 使用 Socket, 使得不同的主机之间可以进行通信, 从而实现数据交换。Socket 通信可用于在双方建立起连接后直接进行数据的传输, 还可以在连接时实现信息的主动推送, 而不需要每次都由客户端向服务器发送请求。Socket 的主要特点包括数据丢失率低, 使用简单且易于移植等。

Socket 在工作的时候会将进行连接的两端分成服务器端和客户端, 服务器程序将在一个众所周知的端口上监听服务请求, 换句话说, 就是服务进程始终是存在的, 直到有客户端的访问请求唤醒服务器端进程为止, 此时, 服务器端进程会和客户端进程之间进行通信, 交换数据。Socket 服务器端和客户端通信的流程图如图 6-3 所示。

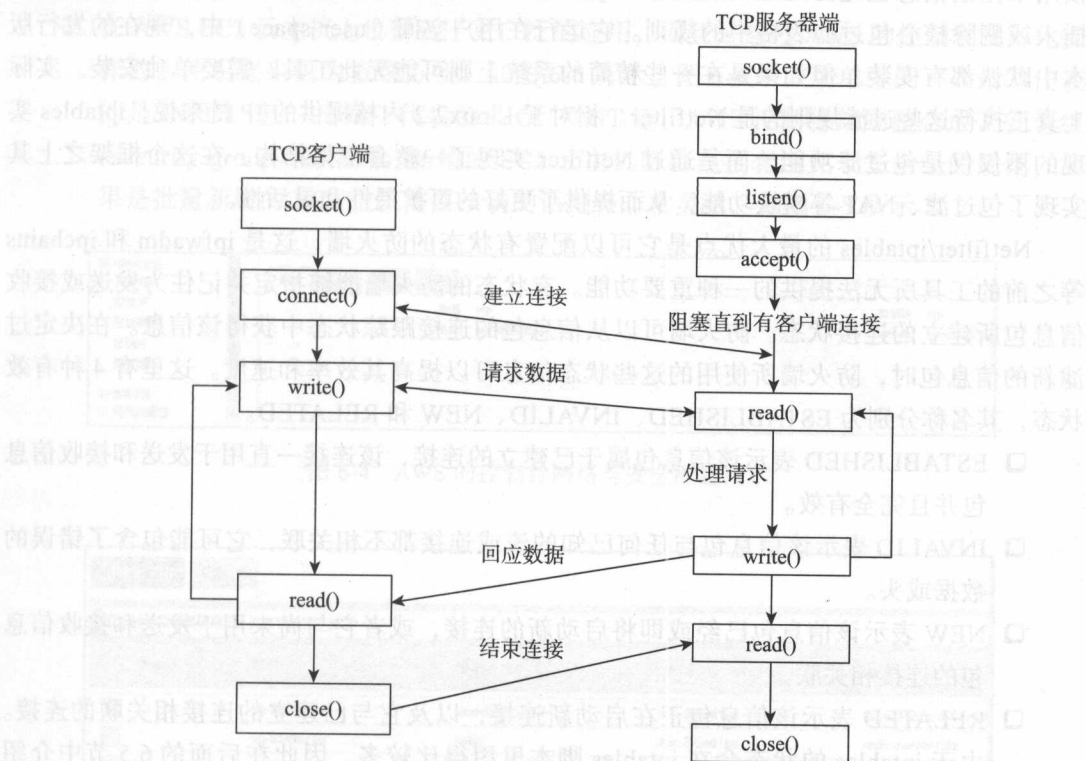


图 6-3 Socket 服务器端和客户端通信过程流程图

关于网络的其他基础知识希望大家也能够有所了解和掌握, 比如子网划分、UDP 的连接原理、硬件防火墙的工作模式等, 如果确实对网络这块不熟悉, 建议先预习下网络基础知识和概念, 这对于我们接下来掌握 Linux 防火墙的知识还是很有帮助的。无论你从事的是系统工作还是开发工作, 基础的网络知识都是必不可少的, 推荐大家有时间阅读一下基

础网络知识，限于篇幅，本书就不做详细说明了。

6.2 Linux 防火墙的概念

Linux 防火墙其实并不是特别专业的叫法，我们所说的 Linux 防火墙其实指的是 Linux 下的 Netfilter/iptables。Netfilter/iptables 是 Linux 内核集成的 IP 信息包过滤系统。

虽然 Netfilter/iptables IP 信息包过滤系统可当作一个整体来看待，但其实它们是该过滤系统的两个组件，Netfilter 是内核的模块实现，iptables 是上层的操作工具。Netfilter 是 Linux 核心中的一个通用架构，运行在内核空间（kernel space）。iptables 提供了一系列的表（tables），每个表都由若干个链（chains）组成，而每条链中可以由一条或数条规则（rule）组成（我们常用的是其中的三表五链），其规则又是由一些信息包过滤表组成，这些表包含内核用来控制信息包过滤处理的规则集。iptables 是一个管理内核包过滤的工具，可以加入、插入或删除核心包过滤表格中的规则。它运行在用户空间（user space）中，现在的发行版本中默认都有安装，但如果是在一些精简的系统上则可能无此工具，需要单独安装。实际上真正执行这些过滤规则的是 Netfilter。相对于 Linux 2.2 内核提供的 IP 链来说，iptables 实现的不仅仅是包过滤功能，而是通过 Netfilter 实现了一整套框架结构，在这个框架之上其实现了包过滤、NAT 等模块功能，从而提供了更好的可扩展性和灵活性。

Netfilter/iptables 的最大优点是它可以配置有状态的防火墙，这是 ipfwadm 和 ipchains 等之前的工具所无法提供的一种重要功能。有状态的防火墙能够指定并记住为发送或接收信息包所建立的连接状态。防火墙可以从信息包的连接跟踪状态中获得该信息。在决定过滤新的信息包时，防火墙所使用的这些状态信息可以提高其效率和速度。这里有 4 种有效状态，其名称分别为 ESTABLISHED、INVALID、NEW 和 RELATED。

- ❑ ESTABLISHED 表示该信息包属于已建立的连接，该连接一直用于发送和接收信息包并且完全有效。
 - ❑ INVALID 表示该信息包与任何已知的流或连接都不相关联，它可能包含了错误的的数据或头。
 - ❑ NEW 表示该信息包已经或即将启动新的连接，或者它与尚未用于发送和接收信息包的连接相关联。
 - ❑ RELATED 表示该信息包正在启动新连接，以及它与已建立的连接相关联的连接。
- 由于 iptables 的状态会在 iptables 脚本里用得比较多，因此在后面的 6.5 节中介绍 iptables 基础知识时将会进行详细说明。

Netfilter/iptables 的另一个重要优点是，它使得用户可以完全控制防火墙的配置和信息包过滤，可以通过定制规则来满足自己的特定需求，从而只允许自己想要的网络流量进入系统。

另外，Netfilter/iptables 是免费的，这对于那些想要节省费用的人来说十分理想，它可

以代替昂贵的防火墙解决方案。附带说明一下，iptables 和 Netfilter 的确存在差别，尽管它们经常被用来相互替换使用，Netfilter 是用来实现 Linux 内核中防火墙的 Linux 内核空间程序代码段，它要么被直接编译进内核，要么被包含在模块中。而 iptables 是用来管理 Netfilter 防火墙的用户程序，在这里统一将 Netfilter/iptables 简称为 iptables。

6.3 Linux 防火墙在企业中的应用

Linux 防火墙（即 Netfilter/iptables IP 过滤系统）在企业中的应用非常广泛，那么，它究竟应用到哪些方面了呢？

- 对于 IDC 机房的服务器，可以用 Linux 防火墙来代替硬件机防火墙，由于 IDC 机房的机器一般是没有硬件防火墙的，因此使用开源免费的 iptables 是一个性价比比较高的选择。
- 在 AWS EC2 云主机上也得到了更广泛的应用，而且是通过 AWS 的控制台图形化操作的，使用起来更为直观方便，如图 6-4 所示。图 6-4 中，菜单中的“入站”代表的是 INPUT 链，“出站”代表的是 OUTPUT 链，AWS 控制将规则整合成了安全组（security group）的格式，新增的 EC2 云主机可以直接套用几个安全组的规则，如果是批量新增的 EC2 主机需要改动的话则会非常灵活，如图 6-5 所示。

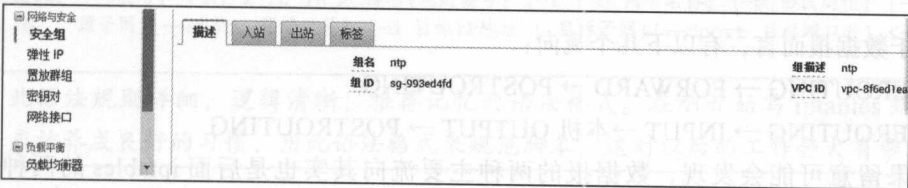


图 6-4 AWS 的控制台网络与安全配置图示

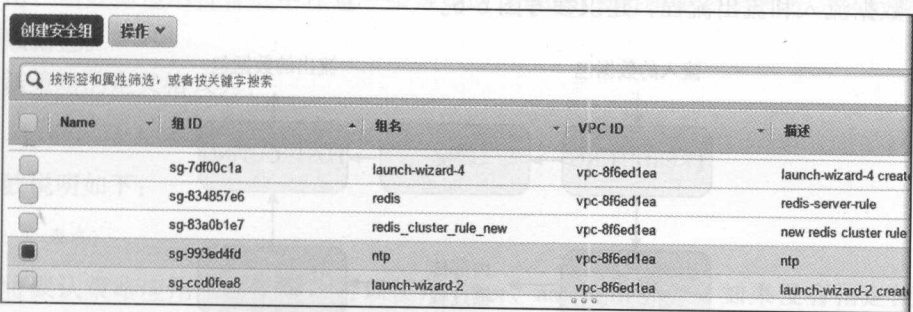


图 6-5 AWS EC2 云主机的安全组创建菜单图示

- 利用 iptables 来作为企业的 NAT 路由器，从而代替传统的路由器供企业内部员工上

网之用，在节约成本和进行有效控制上，Linux 防火墙确实有它独有的优势。

❑ 结合 Squid 作为企业内部上网的透明代理。传统的代理需要在浏览器里配置代理服务信息，而 iptables 结合 Squid 的透明代理则可以把客户端的请求重定向到代理服务器的端口，让客户端感觉不到代理的存在，当然，客户端也无须做任何代理设置。

❑ 用于外网 IP 向内网 IP 映射。我们可以假设有一家 ISP 提供园区 Internet 接入服务，为了方便管理，该 ISP 分配给园区用户的 IP 地址都是内网 IP，但是部分用户要求建立自己的 Web 服务器对外发布信息，这时，可以在防火墙的外部网卡上绑定多个合法 IP 地址，然后通过 IP 映射使发给其中某一个 IP 地址的包转发至内部用户的 Web 服务器上，这样内部的 Web 服务器也就可以对外提供服务了，这种形式的 NAT 一般称为 DNAT。后面的集群架构环节中，经常将负载均衡器的内网 VIP 的 80 和 443 端口通过防火墙映射成公网 IP 的 80 和 443 端口，这也是 DNAT 的实现形式之一。

❑ 防止轻量级的 DOS 攻击，比如 ping 攻击及 SYN 洪水攻击，我们利用 iptables 来做相关安全策略还是很有效果的。

6.4 Linux 防火墙的语法

对于数据报而言，有以下几个流向：

PREROUTING → FORWARD → POSTROUTING

PREROUTING → INPUT → 本机 OUTPUT → POSTROUTING

如果留意可能会发现，数据报的两种主要流向其实也是后面 iptables 的两种工作模式：一是用作 NAT 路由器，另一种是用作主机防火墙，所以对应地要在 iptables 的规则链上做文章（工作中多用于主机防火墙，大家也可以将学习的重心放在这点上）。更为详细的 iptables 数据流入和流出流程，建议参考图 6-6。

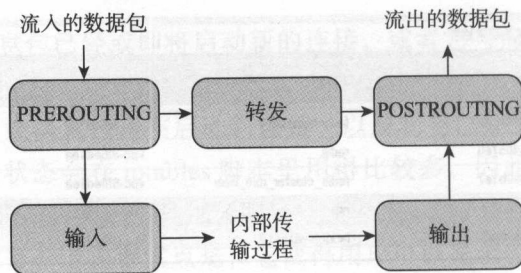


图 6-6 iptables 数据包流入和流出详细流程图

iptables 会根据不同的数据处理包处理功能使用不同的规则表。它包括如下 3 个表：

filter、nat 和 mangle。


- ❑ filter 是默认的表，包含真正的防火墙过滤规则。内建的规则链包括：INPUT、OUTPUT 和 FORWARD。
- ❑ nat 表包含源、目的地址及端口转换使用的规则，内建的规则链包括 PREROUTING、OUTPUT 和 POSTROUTING。
- ❑ mangle 表包含用于设置特殊的数据包路由标志的规则。随后 filter 表中的规则会检查这些标志。内建的规则链包括：PREROUTING、INPUT、FORWARD、POSTROUTING 和 OUTPUT。

表中对应的相关规则链的功能如下。

- ❑ INPUT 链：当一个数据包由内核中的路由计算确定为本地的 Linux 系统后，它会通过 INPUT 链的检查。
- ❑ OUTPUT 链：保留给系统自身生成的数据包。
- ❑ FORWARD 链：经过 Linux 系统路由的数据包（即当 iptables 防火墙用于连接两个网络时，两个网络之间的数据包必须流经该防火墙）。
- ❑ PREROUTING 链：用于修改目的地址（DNAT）。
- ❑ POSTROUTING 链：用于修改源地址（SNAT）。

iptables 详细语法如下所示：

```
iptables [-t 表名] [-A | I | D | R > 链名 [规则编号] [-i | o 网卡名称] [-p 协议类型] [-s 源IP地址 | 源子网] [--sport 源端口号] [-d 目标IP地址 | 目标子网] [--dport 目标端口号] [-j 动作>
```

 **注意** 此语法规则详细，逻辑清晰，推荐记忆此语法格式。在刚开始写 iptables 规则时就应该养成良好的习惯，用此语法格式来规范脚本，这对以后的工作会大有帮助。

下面是关于语法的详细说明。

(1) 定义默认策略

作用：当数据包不符合链中任意一条规则时，iptables 将根据该链预先定义的默认策略来处理数据包。

默认策略的定义格式为：

```
iptables [-t 表名] <-P> <链名> <动作>
```

参数说明如下：

[-t 表名]

指将默认策略应用于哪个表，可以使用 filter、nat 和 mangle，如果没有指定使用哪个表，iptables 就默认使用 filter 表。

<-P>

定义默认策略。

<链名>

指将默认策略应用于哪个链，可以使用 INPUT、OUTPUT、FORWARD、PREROUTING 和 POSTROUTING。

<动作>

处理数据包的动作，可以使用 ACCEPT（接受数据包）和 DROP（丢弃数据包）。

(2) 查看 iptables 规则

查看 iptables 规则的命令格式为：

```
iptables [-t 表名] <-L> [链名]
```

参数说明如下：

[-t 表名]

指查看哪个表的规则列表，表名可以使用 filter、nat 和 mangle，如果没有指定使用哪个表，iptables 就默认查看 filter 表的规则列表。

<-L>

查看指定表和指定链的规则列表。

[链名]

指查看指定表中哪个链的规则列表，可以使用 INPUT、OUTPUT、FORWARD、PREROUTING 和 POSTROUTING，如果不指明哪个链，则将查看某个表中所有链的规则列表。

(3) 增加、插入、删除、替换 iptables 规则

参数说明如下：

[-t 表名]

定义将默认策略应用于哪个表，可以使用 filter、nat 和 mangle，如果没有指定使用哪个表，iptables 就默认使用 filter 表。

-A

新增加一条规则，该规则将会增加到规则列表的最后一行，该参数不能使用规则编号。

-I

插入一条规则，原本该位置上的规则将会往后顺序移动，如果没有指定规则编号，则在第一条规则前插入。

-D

从规则列表中删除一条规则，可以输入要删除的完整规则，或者直接指定规则编号加以删除。

-R

替换某条规则，规则被替换并不会改变顺序，必须要指定被替换的规则编号。

<链名>

指定查看表中哪个链的规则列表，可以使用 INPUT、OUTPUT、FORWARD、PREROUTING 和 POSTROUTING。

[规则编号]

规则编号在插入、删除和替换规则时使用，编号是按照规则列表的顺序排列的，规则列表中第一条规则的编号为 1。

[-i | o 网卡名称]

i 是指定数据包从哪块网卡进入，o 是指定数据包从哪块网卡输出。网卡名称可以使用 ppp0、eth0 和 eth1 等。

[-p 协议类型]

可以指定规则应用的协议，包含 TCP、UDP 和 ICMP 等。

[-s 源 IP 地址 | 源子网]

-s 后面接源主机的 IP 地址或子网地址。

[--sport 源端口号]

--sport 后面接数据包的 IP 源端口号。

[-d 目标 IP 地址 | 目标子网]

-d 后面接目标主机的 IP 地址或子网地址。

[--dport 目标端口号]

--dport 后面接数据包的 IP 目标端口号。

<-j 动作>

下面是处理数据包的动作，以及各个动作的详细说明。

- ❑ ACCEPT：接收数据包。
- ❑ DROP：丢弃数据包。
- ❑ REDIRECT：将数据包重新转向到本机或另一台主机的某一个端口，通常实现透明代理或对外开放内网的某些服务。
- ❑ REJECT：拦截该数据封包，并发回封包通知对方。
- ❑ SNAT：源地址转换，即改变数据包的源地址。例如：将局域网的 IP（10.0.0.1/24）转化为广域网的 IP（203.93.236.141/24），在 NAT 表的 POSTROUTING 链上进行该动作。
- ❑ DNAT：目标地址转换，即改变数据包的目标地址。例如：将广域网的 IP（203.93.236.

141/24) 转化为局域网的 IP (10.0.0.1/24), 在 NAT 表的 PREROUTING 链上进行该动作。

❑ MASQUERADE: IP 伪装, 即常说的 NAT 技术, MASQUERADE 只能用于 ADSL 等拨号上网的 IP 伪装, 也就是主机的 IP 是由 ISP 动态分配的, 如果主机的 IP 地址是静态固定的, 就要使用 SNAT。

❑ LOG: 日志功能, 将符合规则的数据包的相关信息记录在日志中, 以便管理员的分析和排错。

(4) 清除规则和计数器

在新建规则时, 往往需要清除原有的旧规则, 以免它们影响新设定的规则。如果规则比较多, 逐条删除就会十分麻烦, 这时可以使用 iptables 提供的清除规则参数达到快速删除所有的规则的目的。

定义参数的格式为:

```
iptables [-t 表名] <-F | -Z>
```

参数说明如下:

[-t 表名]

指定将默认策略应用于哪个表, 可以使用 filter、nat 和 mangle, 如果没有指定使用哪个表, iptables 就默认使用 filter 表。

-F

通过如下命令删除指定表中的所有规则。

-Z

将指定表中的数据包计数器和流量计数器归零。

6.5 iptables 的基础知识

6.5.1 iptables 的状态 state

6.2 节里提到过 state 这个定义, 这里将解释一下 iptables 防火墙的状态 (state)。

比如, 当我们用 PuTTY 远程工具访问远程主机的 SSH 端口时, 主机和远程主机进行通信。此时, 静态的防火墙会这样处理:

检查进入机器的数据包, 发现数据的来源是 22 端口, 当这些数据包被允许时进入主机本身, 连接之后相互通信的数据也一样, 检查每个数据, 如果发现数据包来源于 22 端口, 则允许通过。

如果使用有状态的防火墙将如何处理呢?

在连接远程主机成功之后, 主机把这个连接记录下来, 当有数据从远程 SSH 服务器

进入你的机器时，它会检查自己的连接状态表，如果发现这个数据来源于一个已经建立连接，则允许这个数据包进入。

以上两种处理方法中，很明显静态防火墙比较生硬，而 iptables 防火墙则相对智能一些，这也是 iptables 防火墙的特点之一。

下面将解释以下几种 state 状态。

- ❑ NEW：如果你的主机向远程机器发出一个连接请求，这个数据包的状态就是 NEW。
- ❑ ESTABLISHED：在连接建立之后（完成 TCP 的三次握手后），远程主机和你的主机通信数据的状态为 ESTABLISHED。
- ❑ RELATED：和现有联机相关的新联机封包。像 FTP 这样的服务，用 21 端口传送命令，而用 20 端口（port 模式）或其他端口（PASV 模式）传送数据。在已有的 21 端口上建立好连接后发送命令，用 20 或其他端口传送的数据（FTP-DATA），其状态是 RELATED。
- ❑ INVALID：无效的数据包，不能被识别属于哪个连接或没有任何状态，通常这种状态的数据包会被丢弃。

有了以上基础知识后，接下来就可以进行一个简单的实验了。

首先，还是来设置默认规则，命令如下所示。

```
iptables -P INPUT DROP
```

这样机器会将进入主机的所有数据都丢弃掉，建议大家写 iptables 脚本时首先默认禁止一切连接，然后再根据应用或需求开放相应的端口。

如果有一台主机只用于个人桌面应用，也就是此主机不提供任何服务，那么，就可以禁止其他的机器向该主机发送任何连接请求，命令如下：

```
iptables -A INPUT -m state --state NEW -j DROP
```

这个规则是将发送到该主机的所有数据包（状态是 NEW 的包）全部丢弃。也就是不允许其他的机器主动发起对该主机的连接，但是该主机却可以主动连接其他的机器，不过仅仅是连接而已，连接之后的数据就是 ESTABLISHED 状态的。这时，再加上下面这一条语句，其作用是允许所有已经建立连接，或者与之相关的数据通过：

```
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

现在根据上面的语句编写一个简单的 iptables 脚本，作为个人桌面主机的防火墙，脚本如下：

```
#!/bin/bash
iptables -F
iptables -F -t nat
iptables -X
iptables -Z
iptables -P INPUT DROP
iptables -A INPUT -m state --state NEW -j DROP
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```


给此脚本 x 权限，命令如下：

```
chmod +x iptables.sh
```

前面几条语句是将其默认规则全部清除掉，让此脚本后面的语句生效。

其实，第二条可以被注释掉，那一条规则完全可以省去，让默认规则处理即可。

是不是很简单呢？对于个人桌面应用来说，只需要用刚才介绍的那两条语句，就能让你接入 Internet 网的主机足够安全。而且可以随意访问 Internet，但是外部却不能主动发起对你机器的连接。

可以看到，有状态的防火墙比静态防火墙要“智能”一些，而且规则的设置也容易一些。

执行以上脚本后，查看一下 iptables 规则，命令如下：

```
iptables -nv -L
```

命令显示结果如下：

```
Chain INPUT (policy DROP 0 packets, 0 bytes)
pkts bytes target      prot opt in      out     source        destination
0      0 DROP          all  --  *       *       0.0.0.0/0     0.0.0.0/0     state NEW
37    2520 ACCEPT       all  --  *       *       0.0.0.0/0     0.0.0.0/0     state
RELATED, ESTABLISHED
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target      prot opt in      out     source        destination
Chain OUTPUT (policy ACCEPT 1532 packets, 1224K bytes)
pkts bytes target      prot opt in      out     source        destination
```

另外，在执行以上脚本后，我们会发现此机器会拒绝一切的数据接入，但是原先的 SSH 并没有被断开？这是为什么呢，这是因为“iptables -A INPUT -m state --state ESTABLISHED, RELATED -j ACCEPT”这段代码发挥了作用，我们原先建立的连接还存在，所以主机不会将此 ESTABLISHED 连接断掉，state 的优势在这里发挥得淋漓尽致。



注意 以上脚本在实验环境下尝试即可，切勿应用于生产服务器，因为默认会拒绝一切连接的。

6.5.2 iptables 的 conntrack 记录

先来看看怎样阅读 /proc/net/nf_conntrack 里的 conntrack 记录。这些记录表示的是当前被跟踪的连接。如果安装了 nf_conntrack 模块，就可以查看 nf_conntrack 记录了，命令如下：

```
cat /proc/net/nf_conntrack
```

命令显示结果如下：

```
ipv4      2 tcp      6 431999 ESTABLISHED src=192.168.1.204 dst=192.168.1.11
sport=22 dport=50233 src=192.168.1.11 dst=192.168.1.204 sport=50233 dport=22
```

```

[ASSURED] mark=0 secmark=0 use=2
ipv4      2 tcp          6 431993 ESTABLISHED src=192.168.1.211 dst=192.168.1.204
sport=41039 dport=80 src=192.168.1.204 dst=192.168.1.211 sport=80 dport=41039
[ASSURED] mark=0 secmark=0 use=2
ipv4      2 udp          17 26 src=0.0.0.0 dst=255.255.255.255 sport=68 dport=67
[UNREPLIED] src=255.255.255.255 dst=0.0.0.0 sport=67 dport=68 mark=0
secmark=0 use=2

```


conntrack 模块维护的所有信息都包含在这个例子当中了，通过它们就可以知道某个特定的连接处于什么状态。首先显示的是协议，这里是 TCP，接着是十进制的 6（TCP 的协议类型代码是 6）。之后的 117 是这条 conntrack 记录的生存时间，它会有规律地被消耗掉，直到收到这个连接的更多包。那时，这个值就会被设定为当时那个状态的默认值。接下来就是这个连接在当前时间点的状态。上面的例子说明了这个包处在状态 SYN_SENT 下，这个值是 iptables 显示的，以便我们好理解，而内部用的值稍有不同。SYN_SENT 说明我们正在观察的这个连接只在一个方向发送了一个 TCP SYN 包。再下面是源地址、目的地址、源端口和目的端口。其中有个特殊的词 UNREPLIED，说明这个连接还没有收到任何回应。最后，是希望接收的应答包的信息，它们的地址和端口与前面是相反的。

当一个连接在两个方向上都有传输时，conntrack 记录就会删除 [UNREPLIED] 标志，然后重置。末尾有 [ASSURED] 的记录说明两个方向已没有流量。这样的记录是确定的，在连接跟踪表装满时，是不会被删除的，没有 [ASSURED] 的记录就要被删除。连接跟踪表能容纳多少条记录是被一个变量控制的，它可由内核中的 ip_sysctl 函数设置。默认值取决于你的内存大小，128MB 可以包含 8 192 条目录，256MB 是 16 376 条，如果在生产服务器上通过加载模块的方法开启了 nf_conntrack 功能，就要注意内存方面的使用情况，此模块是极消耗内存的，对系统性能影响非常大，而且极容易发生以下错误：

```
nf_conntrack: table full, dropping packet
```

具体原因是线上的机器启用了 nf_conntrack 模块以后，服务器的连接数太大，内核的连接跟踪系统（Connection Tracking System）没有足够的空间来存放连接的信息，解决方法就是调整内核参数来增大这个空间。

基于以上种种原因，除了特殊原因以外，不建议在线上服务器上开启 iptables 的 conntrack 功能。

 **注意** 老版的 iptables 的 conntrack 称为 ip_conntrack，新版的名称为 nf_conntrack。nf_conntrack 支持 IPv4 和 IPv6，而 ip_conntrack 只支持 IPv4。

6.5.3 关于 iptables 模块的说明

大多数 Linux 版本实现 iptables 时会使用一系列可载入的程序模块，几乎所有的模块在第一次使用时都会自动动态载入，当然了，我们在撰写 iptables 脚本时也可以通过 modprobe 有选择地载入模块，示例如下：

```
modprobe ipt_MASQUERADE
modprobe nf_conntrack_ftp
modprobe nf_nat_ftp
```

新版的 iptables 有一点很智能，对于以前的一些老模块（比如 ip_nat_ftp 和 ip_conntrack_ftp）也能载入，并且会自动更改模块名称，可以用如下命令来查看：

```
lsmod | grep ip
```

命令显示结果如下所示：

```
ipt_MASQUERADE      2466  0
nf_nat              22759  2 nf_nat_ftp,ipt_MASQUERADE
ipt_LOG             5845  1
nf_conntrack_ipv4   9506  3 nf_nat
nf_defrag_ipv4      1483  1 nf_conntrack_ipv4
nf_conntrack        79357  6 nf_nat_ftp,nf_conntrack_ftp,ipt_MASQUERADE,nf_
    nat,nf_conntrack_ipv4,xt_state
iptable_filter      2793  1
ip_tables           17831  1 iptable_filter
```

现在很多朋友都喜欢自己开发新的模块来实现更为强大的功能，这个问题不是本书的重点，有兴趣的朋友可以自行开发和测试。

6.5.4 iptables 防火墙初始化的注意事项

在与一些系统管理员朋友线下交流时，笔者发现大家在操作 iptables 防火墙时经常遇到的一个问题就是：有时误操作 iptables 而将自己也拦截在机器之外，如果没有 KVM 切换器的话就只有去机房重启机器了。其实这个问题是有解决办法的，在此特推荐给大家。

可以先配置一个 crontab 计划任务，每 5 分钟运行一次，脚本内容如下：

```
*/5 * * * * /etc/init.d/iptables stop
```

这样即使你的脚本存在错误设置的（或丢失）规则时，也不至于将你锁在计算机外而无法返回与计算机的连接，这样你就可以放心大胆地调试你的脚本了。鉴于许多读者在学习及调试 iptables 脚本时也是使用的托管 IDC 机房，所以推荐采用此方法。

6.5.5 如何保存运行中的 iptables 规则

使用 iptables-save 和 iptables-restore 的一个最重要的原因是，它们能在相当大的程度上提高装载、保存规则的速度。使用脚本更改规则的一个问题是，改动每个规则都要调用命令 iptables，而每一次调用 iptables，首先都要把 Netfilter 内核空间中的整个规则集提取出来，然后再插入或附加，或者做其他的改动，最后，再把新的规则集从它的内存空间插入到内核空间中，这会花费很多时间。

为了解决这个问题，可以使用命令 iptables-save 和 iptables-restore。iptables-save 用来把规则集保存到一个特殊格式的文本文件里，而 iptables-restore 则用来把这个文件重新装

入内核空间中。这两个命令最好的地方在于只需要调用一次就可以装载和保存规则集，而不像在脚本中每个规则的改动都要调用一次 iptables。iptables-save 运行一次就可以把整个规则集从内核里提取出来，并保存到文件里，而 iptables-restore 每次只会装入一个规则表。换句话说，对于一个很大的规则集，如果用脚本来设置，那么这些规则就会反反复复地被卸载、安装，而我们现在可以把整个规则集一次性保存下来，安装时一次一个规则表，这样就可以节约大量的时间了。如果你的工作对象是一组巨大的规则，采用这两个工具将是明智的选择。

系统启动 iptables 的规则后，默认有如下规则（虽然比较人性化，但很多时候达不到我们的要求，所以这里大家了解一下就好，不需要做深入研究，可以用 cat 命令来查看）：

```
cat /etc/sysconfig/iptables
```

命令显示结果如下所示：

```
# Firewall configuration written by system-config-securitylevel
# Manual customization of this file is not recommended.
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:RH-Firewall-1-INPUT - [0:0]
-A INPUT -j RH-Firewall-1-INPUT
-A FORWARD -j RH-Firewall-1-INPUT
-A RH-Firewall-1-INPUT -i lo -j ACCEPT
-A RH-Firewall-1-INPUT -p icmp --icmp-type any -j ACCEPT
-A RH-Firewall-1-INPUT -p 50 -j ACCEPT
-A RH-Firewall-1-INPUT -p 51 -j ACCEPT
-A RH-Firewall-1-INPUT -p udp --dport 5353 -d 224.0.0.251 -j ACCEPT
-A RH-Firewall-1-INPUT -p udp -m udp --dport 631 -j ACCEPT
-A RH-Firewall-1-INPUT -p tcp -m tcp --dport 631 -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
-A RH-Firewall-1-INPUT -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

目前编写和调试 iptables 防火墙的通用做法还是通过脚本来进行的，这相对而言更为方便，调试的效率也更高，特别是按照标准流程来编写 iptables 脚本以后。当然这两种方法各有各的优点，我们可以根据自己的环境来选择到底采用哪种方法来保存 iptables 脚本，笔者个人还是倾向于自己手动编写 iptables 脚本。

6.6 如何流程化编写 iptables 脚本

（1）根据需求调整系统内核

例如 TCP 的 SYN 缓冲（syncookies）是一种快速检测和防御 SYN 洪水攻击的机制，如

下的命令可以启用 SYN 缓冲：

```
echo "1" > /proc/sys/net/ipv4/tcp_syncookies
```

另外，如果以 iptables 作为 NAT 路由器，对于存在着多个网卡的情况，则要开启 IP 转发功能，用于多网卡之间数据的流通，命令如下：

```
echo "1" > /proc/sys/net/ipv4/ip_forward
```

其他适用于 iptables 防火墙的内核调整可以根据需求自行设定。

(2) 加载 iptables 模块

由于这里不是以服务的方式启动 iptables 的，而是采用 service 的方式，所以需要手动加载 iptables 模块，例如：

```
modprobe ip_tables
modprobe iptable_nat
modprobe ip_nat_ftp
modprobe ip_nat_irc
modprobe nf_conntrack
modprobe ip_conntrack_ftp
modprobe ipt_MASQUERADE
```

接下来可以用 lsmod 来查看加载的模块，命令如下：

```
lsmod | grep "ip"
```

此命令显示结果如下：

```
ipt_MASQUERADE      2466  0
iptable_nat         6158  0
nf_nat              22759  4  ipt_MASQUERADE,nf_nat_irc,nf_nat_ftp,iptable_nat
nf_conntrack_ipv4   9506  3  iptable_nat,nf_nat
nf_conntrack        79357  8  ipt_MASQUERADE,nf_nat_irc,nf_conntrack_irc,nf_nat_ftp,nf_
    conntrack_ftp,iptable_nat,nf_nat,nf_conntrack_ipv4
nf_defrag_ipv4      1483  1  nf_conntrack_ipv4
ip_tables           17831  1  iptable_nat
```

新版的 iptables 会自动使用新模块的名称来代替旧模块的名称。

(3) 清空所有的表链规则（包括自定义的）

命令如下：

```
iptables -F
iptables -X
iptables -Z
iptables -F -t nat
iptables -X -t nat
iptables -Z -t nat
iptables -X -t mangle
```

(4) 定义默认策略

一般来说为了搭建安全的防火墙，默认是拒绝一切流量连接的，所以三表五链默认规

则都应该是 DROP，但对于实际线上的 iptables 脚本，建议按如下方式配置（因为一般认为从服务器 OUTPUT 出去的数据和 NAT 出去的数据都是安全的）：

```
iptables -P INPUT DROP
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
iptables -t nat -P PREROUTING ACCEPT
iptables -t nat -P POSTROUTING ACCEPT
iptables -t nat -P OUTPUT ACCEPT
```

(5) 打开“回环”

这样做是为了避免不必要的麻烦，命令如下：

```
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
```

(6) 允许状态为 ESTABLISHED 的数据包进入机器

命令如下：

```
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

(7) 根据需求建立防火墙规则

比如 6.5.1 节介绍的完整的桌面主机防火墙脚本如下：

```
#!/bin/bash
modprobe ip_tables
modprobe iptable_nat
modprobe pf_contrack

iptables -F
iptables -X
iptables -Z
iptables -F -t nat
iptables -X -t nat
iptables -Z -t nat
iptables -X -t mangle

iptables -P INPUT DROP
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
iptables -t nat -P PREROUTING ACCEPT
iptables -t nat -P POSTROUTING ACCEPT
iptables -t nat -P OUTPUT ACCEPT

iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

iptables -A INPUT -m state --state NEW -j DROP
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

此脚本执行后，本来正常提供的 samba 服务马上就断开了，客户端连接此机的 samba

报错，可以查看 iptables 的 conntrack 记录，命令如下：

```
cat /proc/net/pf_conntrack
```

此命令显示结果如下：

```
udp      17 15 src=192.168.1.101 dst=192.168.1.255 sport=137 dport=137 packets=20
bytes=1920 [UNREPLIED] src=192.168.1.255 dst=192.168.1.101 sport=137 dport=137
packets=0 bytes=0 mark=0 secmark=0 use=1
udp      17 12 src=192.168.1.101 dst=192.168.1.255 sport=138 dport=138 packets=1
bytes=269 [UNREPLIED] src=192.168.1.255 dst=192.168.1.101 sport=138 dport=138
packets=0 bytes=0 mark=0 secmark=0 use=1
```

samba 建立连接的 137、138 端口只有一边有流量，TCP 的三次握手被拒，这个肯定是提供不了 samba 服务的，证明此脚本是有效的。

(8) 给脚本 x 权限

给脚本 x 权限后就可以直接执行此脚本了。

大家编写 iptables 脚本时可以参考上述步骤，一步一步地来，这样就不容易出错了，以后熟练了就会习惯成自然。

6.7 学习 iptables 应该掌握的工具

6.7.1 命令行的抓包工具 TCPDump

TCPDump 是入侵分析人员工具包中的一个重要工具。在底层，TCPDump 是一个捕获和分析数据包的软件，也就是说 TCPDump 可以用来监听网络通信，但是实际能够监听到什么样的数据流将取决于所在网络的拓扑结构。它是一款基于命令行的工具，可以通过不同的命令行选项来改变状态，它也提供了丰富的选项可使我们很容易地改变程序的运行方式。在使用 TCPDump 的过程中，我们会发现大部分捕获数据的动作只需要用到一些常用的选项即可，而不需要用到其他全部的选项。另外，TCPDump 存在于绝大多数的 Linux 系统中，是不需要安装即可使用的。下面就以具体的例子来说明它的使用方法。

1) 想要截获 210.27.48.1 的主机收到的和发出的所有数据包，命令如下：

```
tcpdump host 210.27.48.1
```

2) 想要截获主机 210.27.48.1 和主机 210.27.48.2 或 210.27.48.3 的通信，命令如下（在命令行中使用括号时，要用转义符 \ 来对 () 进行转义）：

```
tcpdump host 210.27.48.1 and \(210.27.48.2 or 210.27.48.3 \)
```

3) 想要获取主机 210.27.48.1 和所有主机（除了 210.27.48.2）通信的 IP 包，命令如下：

```
tcpdump ip host 210.27.48.1 and !210.27.48.2
```

4) 想要获取主机 210.27.48.1 接收或发出的 smtp 包, 命令如下:

```
tcpdump tcp port 25 and host 210.27.48.1
```

5) 如果怀疑系统正受到拒绝服务攻击 (DoS), 网络管理员可以通过截获发往本机的所有 ICMP 包, 来确定目前是否有大量的 ping 指令流向服务器, 此时可用如下命令:

```
tcpmdump icmp -n -i eth0
```

6) 想要将其结果生成详细的报告, 命令如下:

```
tcpdump tcp port 25 and host 211.147.1.11 > awstat.txt
```

用 TCPDump 捕获的 TCP 包的输出信息一般是:

```
src > dst: flags data-seqno ack window urgent options
```

这里说明一下 TCPDump 抓取 TCP 包的情况。

src>dst: 表明从源地址到目的地址, flags 是 TCP 包中的标志信息, S 代表 SYN 标志、F 代表 FIN、P 代表 PUSH、R 代表 RST、“.” 表示没有标记, data-seqno 是数据包中数据的顺序号, ack 是下次期望的顺序号, window 是接收缓存的窗口大小, urgent 表明数据包中是否有紧急指针, options 是选项。

由于所涉及的服务器协议大部分是 TCP 协议的, 因此这里只介绍 TCP 包的输出信息。至于 UDP 和 ICMP 协议信息, 可以根据上面介绍的 TCPDump 语法自行研究, 限于篇幅, 这里不做详细说明。

6.7.2 图形化抓包工具 Wireshark

Wireshark 是世界上最流行的网络分析工具。这个强大的工具可以捕捉网络中的数据, 并为用户提供关于网络 and 上层协议的各种信息。与很多其他的网络工具一样, Wireshark 也是使用 pcap 这个网络库来进行封包捕捉的, Wireshark 的前身是 Ethereal, 网络管理员使用 Wireshark 来检测网络问题, 网络安全工程师使用 Wireshark 来检查资讯安全相关问题, 开发者使用 Wireshark 来为新的通信协定除错, 普通使用者使用 Wireshark 来学习网络协定的相关知识。Wireshark 不是入侵侦测软件 (Intrusion Detection Software, IDS), 对于网络上的异常流量行为, 它不会产生警示或任何提示。然而, 深入分析 Wireshark 撷取的封包能够帮助使用者对于网络行为有更清楚的了解。Wireshark 不会对网络封包的内容进行修改, 它只会反映出目前流通的封包资讯。Wireshark 本身也不会将封包送出至网络上。

Wireshark 在 CentOS 6.4 x86_64 下的安装极为方便, 首先, 准备一台安装了图形化界面的机器 (因为 Wireshark 是基于图形化的工具), 然后执行如下命令:

```
yum -y install wireshark
```

之后在命令下面用 XShell4.0 登入, 直接输入命令 wireshark 即可。

下面以一个简单的例子来说明一下, 假设客户端是 192.168.1.100, 用 XShell4.0 连接

到了 192.168.1.101 的服务器端上，端口为 22，那么，如何使用 Wireshark 来进行抓包工作呢？步骤如下。

1) 在服务器端上执行 Wireshark，打开此工具，如图 6-7 所示。

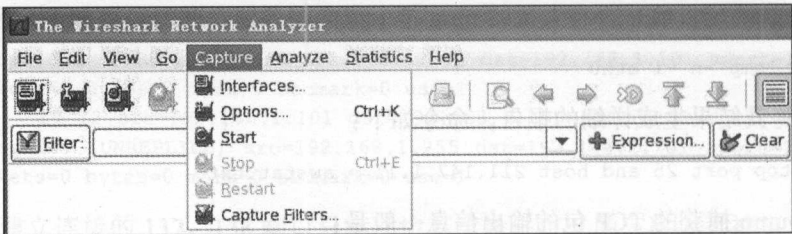


图 6-7 Wireshark 工作界面图

2) 选择“Capture”的“options”选项，在“Capture Filter”里面输入抓包规则，命令如下：

```
host 192.168.1.100 and (host 192.168.1.101 and port 22)
```

Wireshark 语法跟 TCPDump 非常接近，如图 6-8 所示。

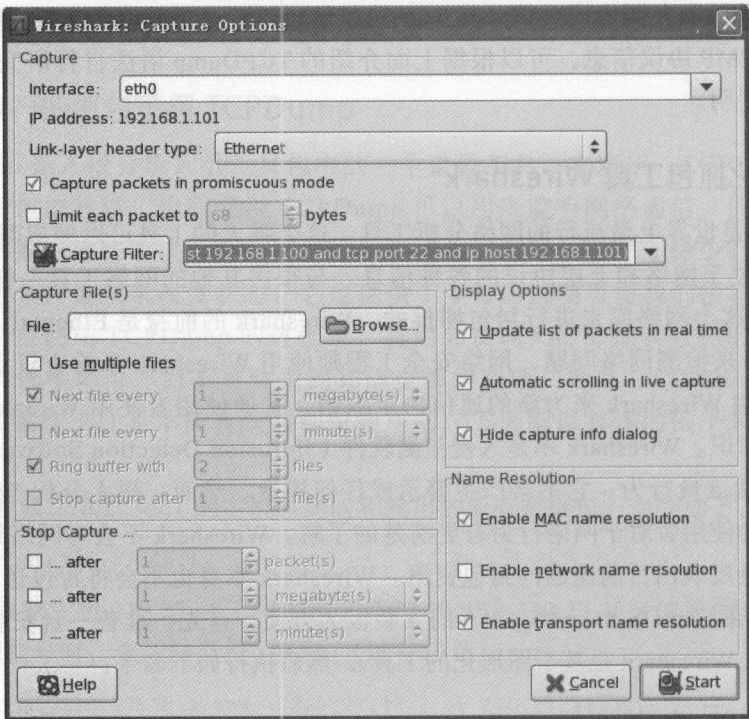


图 6-8 编写 Wireshark 工具抓包规则

3) 然后点击“start”，就可以开始抓包了，如图 6-9 所示。

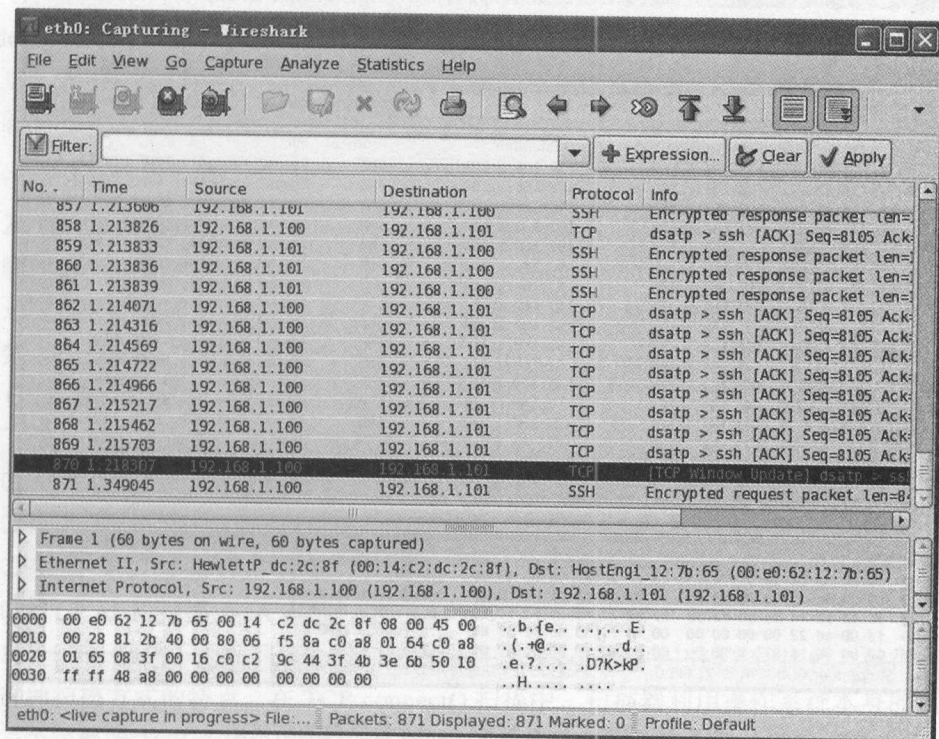


图 6-9 Wireshark 工具抓包结果

抓包结果由大家自行分析，在实际工作中大家可以根据当前服务器的实际状态来选择采用何种抓包工具，Wireshark 是有图形化界面的，TCPDump 则是没有图形化界面的，建议以上两种工具都要掌握，因为它们的语法规则基本是类似的。有兴趣的朋友可以直接在规则里输入 icmp（这是用来运行 ping 协议的），然后在另外的机器上 ping 通服务器端，这样就可以更直观地看到数据的走向了，这点对于以后编写 iptables 脚本大有帮助。这里跟大家交流一下 Wireshark 的一个使用经验，如果我们不知道某项服务要用到哪些协议、哪些端口，可用 Wireshark 抓下包。比如想知道 samba 到底占用了哪个端口，就可以尝试使用 Wireshark，然后得出正确的结论，这样是不是很方便呢？不过，大家写抓包规则时，要排除掉 22 的干扰，提示一下规则，命令如下：

```
host 192.168.1.100 and (host 192.168.1.101 and port not 22)
```

抓包结果如图 6-10 所示，通过对数据包的分析，可得出如下结论：samba 服务占用的端口为 445，所以只需要在 iptables 防火墙上开放 445 端口即可，命令如下：

```
iptables -A INPUT -p tcp --dport 445 -j ACCEPT
```

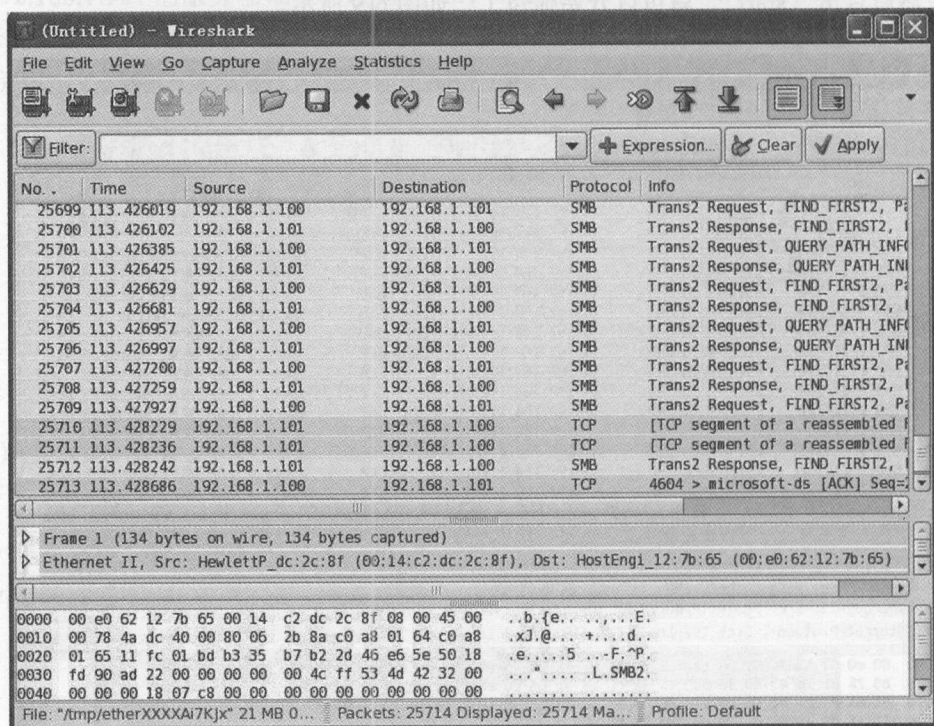



图 6-10 Wireshark 抓取 samba 数据包的结果图

6.7.3 强大的命令行扫描工具 Nmap

系统管理员使用 Nmap 查看一个大的网络系统有哪些主机，以及其上运行了何种服务。它支持多种协议的扫描，如 UDP、TCP connect()、TCP SYN (half open)、ftp proxy (bounce attack)、Reverse-ident、ICMP (ping sweep)、FIN、ACK sweep、Xmas Tree、SYN sweep 和 Null 扫描等。Nmap 还提供了一些实用的功能，比如通过 TCP/IP 来甄别操作系统的类型、秘密扫描、动态延迟和重发、平行扫描，通过并行的 ping 侦测下属的主机、欺骗扫描、端口过滤探测、直接的 RPC 扫描、分布扫描、灵活的目标选择及端口的描述。它的扫描功能异常强大，以至于人们叫它扫描之王。

1. 安装 Nmap

安装 Nmap 要用到一个名为“Windows 包捕获库”的驱动程序 WinPcap——如果你经常从网上下载流媒体电影，可能已经很熟悉这个驱动程序了——某些流媒体电影的地址是加密的，侦测这些电影的真实地址就要用到 WinPcap。WinPcap 的作用是帮助调用程序（即这里的 Nmap）捕获通过网卡传输的原始数据。WinPcap 的最新版本在 <http://netgroup-serv.polito.it/winpcap> 上可以下载到，可支持 Windows 全系列操作系统，下载得到的是一

个执行文件，双击安装，逐步确认使用默认设置就可以了，安装好之后需要重新启动。除了命令行版本之外，www.insecure.org 还提供了一个带 GUI 的 Nmap 版本。和其他常见的 Windows 软件一样，GUI 版本需要安装，该版本的功能基本上和命令行版本一样，鉴于许多人更喜欢用命令行版本，本文后面的说明就以命令行版本为主。而在 CentOS 6.4 x86_64 下安装 Nmap 就简单多了，直接用如下命令即可：

```
yum -y install nmap
```

2. 常用的扫描类型

解开 Nmap 命令行版的压缩包之后，进入 Windows 的命令控制台，再转到安装 Nmap 的目录（如果经常要用 Nmap，最好把它的路径加入到 PATH 环境变量中）。不带任何命令运行 Nmap，Nmap 显示出命令语法，Linux 下是 `nmap --help`（以下命令行操作均适用于 CentOS、FreeBSD 和 Windows 系列）。

下面是 Nmap 支持的 4 种最基本的扫描方式：

- ❑ TCP connect() 端口扫描（-sT 参数、-sP 用于扫描整个局域网段）。
- ❑ TCP 同步（SYN）端口扫描（-sS 参数）。
- ❑ UDP 端口扫描（-sU 参数）。
- ❑ TCP ACK 扫描（-sA 参数）。

TCP SYN 扫描不太好理解，但如果将它与 TCP connect() 扫描进行比较，就很容易可以看出这种扫描方式的特点。在 TCP connect() 扫描中，扫描器利用操作系统本身的系统调用打开一个完整的 TCP 连接，也就是说，扫描器打开了两个主机之间的完整握手过程（SYN、SYN-ACK 和 ACK）。一次完整执行的握手过程表明远程主机端口是打开的。TCP SYN 扫描创建的是半打开的连接，它与 TCP connect() 扫描的不同之处在于，TCP SYN 扫描发送的是复位标记（RST）而不是结束 ACK 标记（即 SYN、SYN-ACK 或 RST）：如果远程主机正在监听且端口是打开的，则远程主机用 SYN-ACK 应答，Nmap 发送一个 RST；如果远程主机的端口是关闭的，则它的应答将是 RST，此时 Nmap 转入下一个端口。TCP SYN 的扫描速度要超过 TCP connect() 扫描。如果采用默认计时选项，在 LAN 环境下扫描一个主机，ping 扫描耗时不到 10 秒，TCP SYN 扫描需要大约 13 秒，而 TCP connect() 扫描耗时最多，大约需要 7 分钟。需要说明的是，TCP SYN 扫描又叫隐蔽扫描，扫描时可隐藏自身 IP，因为它很少在目标机上留下记录，三次握手的过程从来都不会完全实现。

3. 命令行参数说明

Nmap 支持丰富、灵活的命令行参数。例如，如果要扫描 192.168.7 网络，可以用 192.168.7.x/24 或 192.168.7.0-255 的形式指定 IP 地址的范围。指定端口范围使用 -p 参数，如果不指定要扫描的端口，Nmap 默认扫描从 1 到 1 024 再加上 nmap-services 列出的端口，`nmap -sS -O 192.168.0.1` 这样的命令可以对此主机进行操作系统识别。

如果要查看 Nmap 运行的详细过程，只需要启用 verbose 模式即可，加上 -v 参数，或

者加上 `-vv` 参数可获得更加详细的信息，命令如下所示：

```
nmap -sS 192.168.7.1-255 -p 20,21,53-110,30000 --v
```

这表示执行一次 TCP SYN 扫描，启用 verbose 模式，要扫描的网络是 192.168.7，检测 20、21、53 到 110 及 30 000 以上的端口（指定端口清单时中间不要插入空格）。再举一个例子：

```
nmap -sS 192.168.7.1/24 -p 80
```

它将扫描 192.168.7 子网，查找在 80 端口监听的服务器（通常是 Web 服务器）。

有些网络设备，例如路由器和网络打印机，可能会禁用或过滤掉某些端口，从而禁止对该设备或跨越该设备的扫描。初步侦测网络情况时，`-host_timeout<毫秒数>` 参数很有用，它表示超时时间，例如 `nmap-sS host_timeout 10000 192.168.0.1` 命令规定的超时时间是 10 000 毫秒。

网络设备上被过滤掉的端口一般会大大延长侦测时间，设置超时参数有时可以显著降低扫描网络所需要的时间。Nmap 会显示出哪些网络设备响应超时，这时你就可以对这些设备个别处理，从而保证大范围网络扫描的整体速度。当然，`host_timeout` 到底可以节省多少扫描时间，最终还是由网络上被过滤掉的端口数量决定的。

4. 注意事项

也许你对其他端口扫描器比较熟悉，但 Nmap 绝对值得一试。建议先用 Nmap 扫描一个熟悉的系统，感受一下 Nmap 的基本运行模式，待熟悉之后，再将扫描范围扩大到其他系统。首先扫描内部网络看看 Nmap 报告的结果，然后从一个外部 IP 地址开始扫描，注意防火墙、入侵检测系统（IDS）及其他工具对扫描操作的反应。通常，TCP connect() 会引起 IDS 系统的反应，但 IDS 不一定会记录被称为“半连接”的 TCP SYN 扫描。最好将 Nmap 扫描网络的报告整理存档，以便随后参考。

如果你打算了解和使用 Nmap，下面有几个注意事项。

（1）避免误解

不要随意选择扫描目标来测试 Nmap。许多单位把端口扫描视为恶意行为，所以测试 Nmap 最好在内部网络中进行。如有必要，应该告诉同事你正在试验端口扫描，因为扫描可能引发 IDS 警报及其他网络问题。如果不是在你控制的网络、系统及站点上使用该工具，你应该首先查看许可权。记住，尊重他人的网络和系统的隐私意味着别人以后也许也会这样对你。

（2）关闭不必要的服务

根据 Nmap 提供的报告（同时考虑网络的安全要求），关闭不必要的服务，或者调整路由器的访问控制规则（ACL），禁用网络开放给外界的某些端口。

（3）建立安全基准

在 Nmap 的帮助下加固网络，在搞清楚哪些系统和服务可能受到攻击之后，下一步是

从这些已知的系统和服务出发建立一个安全基准，以后如果要启用新的服务或服务器，就可以方便地根据这个安全基准来执行。

6.8 iptables 简单脚本：Web 主机防护脚本

这一节通过编写一个简单的 iptables 脚本来熟悉 iptables 的语法规则。网络拓扑很简单，安装 iptables 的机器 IP 地址为：192.168.1.204，另一台机器的 IP 地址为：192.168.1.200，系统均为 CentOS 6.4 x86_64。

普通的 Web 主机防护脚本比较容易实现，Web 主机主要开放两个端口：80 和 22，其他端口则会关闭，另外由于这里没有涉及多少功能，所以模块的载入也很简单，只涉及了 iptables 的 filter 表的 INPUT 链，所以脚本的初始化也很简单。

可以按照编写 iptables 的流程顺序来编写脚本，脚本内容如下：

```
#/bin/bash
iptables -F
iptables -X
iptables -Z

modprobe ip_tables
modprobe nf_nat
modprobe nf_conntrack

iptables -P INPUT DROP
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT

iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

iptables -A INPUT -p tcp -m multiport --dports 22,80 -j ACCEPT
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
```

这里的 #iptables -P INPUT DROP 符合我们写 iptables 的习惯，即开启 iptables 时默认拒绝一切连接，后面再通过 -A 参数来开放我们需要提供的端口。

iptables 脚本开启后，可以用如下命令查看结果：

```
iptables -nv -L
```

此命令显示结果如下：

```
Chain INPUT (policy DROP 3364 packets, 204K bytes)
pkts bytes target      prot opt in      out     source        destination
 0      0      ACCEPT      all  --  lo      *           0.0.0.0/0     0.0.0.0/0
84 5372      ACCEPT      tcp  --  *       *           0.0.0.0/0     0.0.0.0/0
multiport dports 22,80
0      0 ACCEPT      all  --  *       *           0.0.0.0/0     0.0.0.0/0
```



```
stateRELATED,ESTABLISHED
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target      prot opt in      out     source            destination
Chain OUTPUT (policy ACCEPT 43 packets, 5532 bytes)
pkts bytes target      prot opt in      out     source            destination
0      0 ACCEPT    all  --  *      !o      0.0.0.0/0         0.0.0.0/0
```

iptables 防火墙运行后，将尝试启动此机器的 postfix 服务，打开服务器端口 25，然后通过内网在另外一台机器上 telnet，命令如下：

```
telnet 192.168.1.204 25
```

命令显示结果如下：

```
Trying 192.168.1.204...
telnet: connect to address 192.168.1.204: Connection refused
```

最小化安装的 CentOS 6.4 x86_64 系统默认是没有 Nmap 的，可以通过 yum 命令来进行安装，命令如下：

```
yum -y install nmap
```

这时，在另一台机器上开启 Nmap 扫描，发现 samba 提供服务的端口已经被 iptables 屏蔽了，命令如下：

```
nmap -sT 192.168.1.204
```

此命令显示结果如下：

```
Starting Nmap 5.51 ( http://nmap.org ) at 2015-10-27 23:43 EDT
Nmap scan report for Fabric (192.168.1.204)
Host is up (0.00074s latency).
Not shown: 998 filtered ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
MAC Address: 00:16:3E:05:23:85 (XenSource)
Nmap done: 1 IP address (1 host up) scanned in 4.39 seconds
```

6.9 线上生产服务器的 iptables 脚本

在编写安全的 iptables 脚本之前，依然要先做好准备工作；现在比较新的 CentOS 系统版本是 CentOS 6.4 x86_64，可以通过如下命令查看它自带的 iptables 版本：


```
iptables --version
iptables v1.4.7
```

如果要查看系统的内核版本号，可以用如下命令：

```
uname -r
```


2.6.32-358.el6.x86_64

为什么要采用较新的 CentOS 系统呢,这是因为 iptables 现在有许多新的模块,如果在原有的老系统和老内核上采用这些新的 iptables 模块,则必须要采取重新编译内核的方法。不过现在新版的 CentOS 系统自带的 iptables V1.4.7 已经支持了原先不支持的许多模块,比如 connlimit、recent 模块,所以我们部署 iptables 防火墙时就容易得多了。

 **注意** 鉴于 iptables v1.3.x 和 iptables v1.4.7 的语法有细微的区别,除了线上生产服务器(基于稳定的原则,线上环境不可能随便去更改内核及 iptables 版本)之外,以下所有的 iptables 脚本都以 CentOS 6.4 x86_64 系统、内核版本 2.6.32-358.el6.x86_64、iptables 版本 1.4.7 为平台来说明。

由于这里的服务器都涉及生产服务器,为了防止发生意外事件,所以在调试 iptables 脚本之前,先配置一个 Crontab 计划任务,每 5 分钟关闭一次防火墙,以免将自己千里之外的防火墙 SSH 连接也断掉了,那样就得不偿失了,编辑 /etc/crontab 计划任务的命令如下:

```
* /5 * * * * root /etc/init.d/iptables stop
```

等确保 iptables 万无一失以后,才能清除掉此计划任务。

6.9.1 安全的主机 iptables 防火墙脚本

下面以自己的线上 iRedMail 邮件服务器(此机器系统为 CentOS 5.1 x86_64,此邮件服务器上在线时间比较早,大约在 2008 年左右)为例进行说明,系统的默认策略是 INPUT 为 DROP, OUTPUT、FORWARD 链为 ACCEPT, DROP 设置得比较宽松,因为我们知道出去的数据包比较安全。

脚本代码如下:

```
#/bin/bash
iptables -F
iptables -F -t nat
iptables -X

iptables -P INPUT DROP
iptables -P OUTPUT ACCEPT
iptables -P FORWARD ACCEPT

#load connection-tracking modules
modprobe ip_conntrack
modprobe iptable_nat

iptables -A INPUT -f -m limit --limit 100/sec --limit-burst 100 -j ACCEPT
iptables -A FORWARD -p icmp --icmp-type echo-request -m limit --limit 1/s
--limit-burst 10 -j ACCEPT
iptables -A INPUT -p tcp -m tcp --tcp-flags SYN,RST,ACK SYN -m limit --limit 20/
```

```
sec --limit-burst 200 -j ACCEPT
```

```
iptables -A INPUT -i lo -j ACCEPT
```

```
iptables -A OUTPUT -o lo -j ACCEPT
```

```
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

```
iptables -A INPUT -p tcp -m multiport --dport 80,443,25,465,110,995,143,993,587,465,22 -j ACCEPT
```

查看 iptables 的详细规则，命令如下：

```
iptables -nv -L
```

此命令显示结果如下：

```
Chain INPUT (policy DROP 0 packets, 0 bytes)
pkts bytes target      prot opt in      out     source         destination    limit:
0      0 ACCEPT      all  -f  *      *      0.0.0.0/0      0.0.0.0/0      limit:
avg 100/sec burst 100
0      0 ACCEPT      tcp  --  *      *      0.0.0.0/0      0.0.0.0/0      tcp
flags:0x16/0x02 limit: avg 20/sec burst 200
0      0 ACCEPT      all  --  lo     *      0.0.0.0/0      0.0.0.0/0
26    1925 ACCEPT      all  --  *      *      0.0.0.0/0      0.0.0.0/0
state RELATED,ESTABLISHED
0      0 ACCEPT      tcp  --  *      *      0.0.0.0/0      0.0.0.0/0
multiport dports 80,443,25,465,110,995,143,993,587,465,22
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target      prot opt in      out     source         destination
0      0 ACCEPT      icmp --  *      *      0.0.0.0/0      0.0.0.0/0
icmp type 8 limit: avg 1/sec burst 10
Chain OUTPUT (policy ACCEPT 20 packets, 1873 bytes)
pkts bytes target      prot opt in      out     source         destination
0      0 ACCEPT      all  --  *      lo     0.0.0.0/0      0.0.0.0/0
```

在主机的防护上我们配置了一些安全措施，以防止外部的 ping 和 SYN 洪水攻击，并且考虑到外部的疯狂端口扫描软件可能会影响服务器的入口带宽，所以在这里也做了限制，命令如下：

```
iptables -A INPUT -p tcp --syn -m limit --limit 100/s --limit-burst 100 -j ACCEPT
```

上面的命令每秒钟最多允许 10 个新连接，请注意这里的新连接指的是 state 为 New 的数据包，后面也配置了允许状态为 ESTABLISHED 和 RELATED 的数据通过；另外，阈值（上述命令中为 100）需要根据服务器的实际情况来调整，如果是并发量不大的服务器，则要将这个数值调小，如果是访问量非常大且并发数不小的服务器，则还需要将这个值调大。再来看看以下命令：

```
iptables -A FORWARD -p icmp --icmp-type echo-request -m limit --limit 1/s -limit-burst 10 -j ACCEPT
```

上述命令是为了防止 ping 洪水攻击，限制每秒的 ping 包不超过 10 个。

```
iptables -A INPUT -p tcp -m tcp --tcp-flags SYN,RST,ACK SYN -m limit --limit 20/
```

```
sec --limit-burst 200 -j ACCEPT
```

上面的命令是防止各种端口扫描，将 SYN 及 ACK SYN 限制为每秒钟不超过 200 个，以免将服务器的带宽耗尽。

还可以运行 nmap 工具扫描此机器的公网地址 211.143.1.1（此公网地址已做无害处理），命令如下：

```
nmap -P0 -sS 211.143.1.1
```

此命令的执行结果如下：

```
Starting Nmap 4.11 ( http://www.insecure.org/nmap/ ) at 2009-03-29 16:21 CST
Interesting ports on 211.143.1.1:
```

```
Not shown: 1668 closed ports
```

PORT	STATE	SERVICE
22/tcp	open	ssh
25/tcp	open	smtp
80/tcp	open	http
110/tcp	open	pop3
111/tcp	open	rpcbind
143/tcp	open	imap
443/tcp	open	https
465/tcp	open	smtps
587/tcp	open	submission
993/tcp	open	imaps
995/tcp	open	pop3s
1014/tcp	open	unknown

从结果中可以发现一个 1014 端被某个进程打开了，可用如下命令来查看：

```
lssof -i:1014
```

查看结果可发现又是 rpc.statd 打开的，这个服务每次用的端口都不一样啊！本来想置之不理的，但是如果 rpc.statd 不能正确处理 SIGPID 信号，远程攻击者就会利用这个漏洞关闭进程，进行拒绝服务攻击，所以还是得想办法解决掉这个问题才行。可以看到，rpc.statd 是由服务 nfslock 开启的，进一步查询可得知该服务是一个可选的进程，它允许 NFS 客户端在服务器上对文件进行加锁操作。这个进程对应于 nfslock 服务，于是考虑关掉此服务，命令如下：

```
service nfslock stop
chkconfig nfslock off
```

6.9.2 自动分析黑名单及白名单的 iptables 脚本

本 iptables 脚本是一个自动分析黑名单和白名单的安全脚本，脚本路径为 /root/deny_100.sh。运行此脚本时要注意以下事项：

- ❑ 此脚本能自动过滤掉企业中通过 NAT (Network Address Translation, 网络地址转换) 出去的白名单 IP, 很多中小企业都是以 iptables 作为 NAT 软路由上网的, 可以将一些与我们有业务往来的公司及本公司的安全 IP 添加进白名单, 以防错误过滤。
 - ❑ 这里定义的阈值 DEFINE 是 100, 其实这个值应该根据具体生产环境而定, 50~100 较好。
 - ❑ 此脚本的原理其实很简单, 通过判断瞬间连接数是否大于 100 来抉择, 如果是白名单里的 IP 则跳过; 如果不是, 则用 iptables -I 参数将此恶意 IP 禁掉。这里建议不要用 -A, -A 在 iptables 的规则里是最后添加的, 往往达不到即时剔除的效果。iptables 中针对链的操作编号其实是有规则的, -I 是在规则的最前面插入的, 而 iptables 是按照规则的顺序来生效的, 所以可以采用 -I 实现立即禁止某 IP 的目的。
 - ❑ 此脚本是在线上的邮件服务器上进行调试的, 系统为 CentOS 5.1 x86_64。
- 可以用 cat 命令来查看脚本的内容, 命令如下:

```
cat /root/deny_100.sh
```

/root/deny_100.sh 脚本的内容如下:

```
#!/bin/bash
netstat -an| grep :25 | grep -v 127.0.0.1 |awk '{ print $5 }' | sort|awk -F:
'{print $1,$4}' | uniq -c | awk '$1 >50 {print $1,$2}' > /root/black.txt

for i in `awk '{print $2}' /root/black.txt`
do
COUNT=`grep $i /root/black.txt | awk '{print $1}'`
DEFINE="1000"
ZERO="0"
if [ $COUNT -gt $DEFINE ];
then
grep $i /root/white.txt > /dev/null
if [ $? -gt $ZERO ];
then
echo "$COUNT $i"
iptables -I INPUT -p tcp -s $i -j DROP
fi
fi
done
```

2009 年 3 月 30 日下午 14:25 分, 用下列命令监控:

```
netstat -an| grep :25 | grep -v 127.0.0.1 |awk '{ print $5 }' | sort|awk -F:
'{print $1}' | uniq -c | awk '$1 >100'
```

执行此命令后显示的内容如下:

```
1122 219.136.163.207
17 61.144.157.236
```

219.136.163.207 这个 IP 的瞬间连接数为 1122, 这个值明显不正常, 这极有可能是一

个攻击 IP，用 <http://www.ip138.com> 一查，发现了如下结果：

ip138.com IP查询 (搜索IP地址的地理位置)

您查询的IP:219.136.163.207

本站主数据: 广东省广州市 电信 (荔湾区)


参考数据一: 广东省广州市 电信 (荔湾区)

参考数据二: 广东省广州市荔湾区 电信ADSL

调用 `deny_100.sh` 将此 IP 禁止掉，再运行 `./root/count.sh` 后则再无显示，表明此脚本执行成功，可用 `iptables -nL` 验证，另外，要允许此脚本每 10 分钟执行一次，命令如下：

```
* /10 * * * * root /bin/sh /root/deny_100.sh
```

一般来说，10 分钟或更长时间执行一次此脚本是没有问题的，因此此脚本只要发现有大量可疑的 IP 连接，在排除是白名单的情况下会立即禁止此 IP 访问。

 **注意** 有的朋友喜欢用 `while` 循环的方法，这里也可以用此方法，但要注意防止出现死循环的问题，因此一定要记得带上 `sleep` 语句。

我们可以在局域网内模拟测试攻击来测试此脚本：在本机 `iptables` 的防火墙（Server 机器 IP 为 192.168.1.101）上，将此脚本的监听端口由 25 改成 80，开启 Apache Web 服务。然后在另一台 CentOS 6.4 机器，比如 192.168.1.102 上安装 `Webbench` 软件，用它来模拟 80 端口的攻击，先进入 `/usr/local/src` 目录，然后下载并安装 `Webbench`，命令如下：

```
wget http://blog.s135.com/soft/linux/webbench/webbench-1.5.tar.gz
```

然后解压缩此软件并编译安装，命令如下：

```
tar zxvf webbench-1.5.tar.gz
cd webbench
make && make install
```

如果 `make` 时有如下报错：

```
ctags *.c
/bin/sh: ctags: command not found
make: [tags] Error 127 (ignored)
```

提前安装下 `ctags` 命令即可，如下所示：

```
yum -y install ctags
```

然后用以下命令进行压力测试（模拟端口攻击）：

```
webbench -t 1000 -c 150 http://192.168.1.101
```

此命令行的意思是在 1000s 的单位时间内，模拟 150 个并发去访问 <http://192.168.1.101> 的网站。

我们在服务器上可以执行以下命令，观测 `iptables` 的执行结果：


```

iptables -nv -L
Chain INPUT (policy ACCEPT 849K packets, 57M bytes)
pkts bytes target      prot opt in      out     source      destination
1161 85180 DROP          tcp  --  *      *       192.168.1.102  0.0.0.0/0
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target      prot opt in      out     source      destination
Chain OUTPUT (policy ACCEPT 849K packets, 92M bytes)
pkts bytes target      prot opt in      out     source      destination

```

然后试着在 192.168.1.102 的机器上用 elinks 访问 192.168.1.101 的 Web 服务，这时已经访问不了了，证明此脚本已成功运行，有兴趣的朋友也可以依照以上步骤进行实验。

将此脚本放在线上服务器上使用时，发现对于非 Web 的应用服务器，比如 Mail、DNS 等应用服务器确实很有效果，而对于 Web 应用服务器效果则不是特别明显。这是因为现在的很多 Web 服务器，特别是存储海量图片小文件的服务器，如果单击某一个链接，可能同时会产生多个对应页面的链接，而这个 IP 在 netstat 中对应的链接数就有 100 多个，故而脚本会认为此 IP 是一个危险的 IP，应该过滤掉它，但我们通过 Nginx 的日志分析，发现这个客户端的 IP 是一个正常的客户 IP，因此将此脚本应用于 Web 服务器不太合适。笔者目前也只将其用于邮件服务器和 DNS 服务器，请大家在实际工作中也要注意甄别使用，如果确实需要限制 IP 在单位时间内的连接次数，可以利用 iptables 的 recent 模块来进行，下一节将会跟大家详细说明。

6.9.3 利用 recent 模块限制同一 IP 的连接数

6.9.2 节向大家演示了自动区别黑名单和白名单的 iptables 脚本，发现将其应用于 Web 服务器的效果并不是很好，如果想限制瞬间连接数过大的恶意 IP 地址，可以考虑使用 iptables 的模块 recent。

新版的 iptables 有个实用、简单又高效的功能，可以用它阻止瞬间连接数过多的源 IP。这种阻挡功能在某些地方很受欢迎，比如说某大型讨论区网站，每个网页都有可能遭到无聊人士的连接，一瞬间太多的连接访问将导致服务器呈现呆滞状态。

这时，就需要用到下列三行指令，代码如下：

```

iptables -I INPUT -p tcp --dport 80 -m state --state NEW -m recent --name web --set
iptables -A INPUT -m recent --update --name web --seconds 60 --hitcount 20 -j LOG
--log-prefix 'HTTP attack: '
iptables -A INPUT -m recent --update --name web --seconds 60 --hitcount 20 -j DROP

```

其中，要将 SERVER_IP 换成被攻击的服务器 IP。

第一行：-I 表示将本规则插入到 INPUT 链的最上头。什么样的规则呢？iptables 判断只要是 TCP 性质的联机，目标端口是 80 并且目标 IP 是我们机器 IP 的联机，就可以将这个联机列入这份 Web 清单中。

第二行：-A 表示将本规则附在 INPUT 链的最尾端。如果在 60 秒内，同一个来源连续产生了多个联机，则在到达第 20 个联机时，就对此联机留下 Log 记录，记录行将以 HTTP

attack 开头。

第三行: -A 表示将本规则附在 INPUT 链的最尾端。在与第二行条件相同的情况下,本次的动作则是将此连接给断掉,即将每 60 秒内有 20 个联机的数据包给 DROP 掉。

所以,这三行规则表示,我们允许一个客户端,每一分钟内可以接上服务器的 20 个新连接,具体数值可以由具体的生产环境来决定。这些规则也可以用在 Internet 开放的其他应用服务上,例如 Mail 邮件服务器和 DNS 解析服务器。

为什么新版的 iptables 在阻挡上很有效率呢?因为在旧版的 iptables 中,并没有这些新模块功能,导致我们需要使用操作系统的 Shell (比如 netstat) 接口,周期性地执行网络检查与拦阻动作。前者只动用到网络层的资源,而后者已经是应用层的大量(相对而言)运算。试想一下,服务器资源都已经被非法客户端给消耗殆尽了,哪还有余力周期性地呼叫软件层级的计算,来阻挡非法客户端呢?新版的 iptables 增加了此模块后就可以直接禁止掉恶意的 IP,而不需要调用操作系统的 Shell 接口,所以更有效率。

接下来我们测试一下这个脚本的效果,步骤如下。

1) 将这三句话写成脚本形式,方便执行操作(此测试机器的系统为 CentOS 6.4 x86_64, IP 为 192.168.1.204),代码如下:

```
#!/bin/bash
iptables -I INPUT -p tcp --dport 80 -m state --state NEW -m recent --name web --set
iptables -A INPUT -m recent --update --name web --seconds 60 --hitcount 20 -j LOG
--log-prefix 'HTTP attack: '
iptables -A INPUT -m recent --update --name web --seconds 60 --hitcount 20 -j DROP
```

执行后查看结果,显示如下:

```
Chain INPUT (policy DROP 1 packets, 284 bytes)
pkts bytes target      prot opt in      out     source        destination
0      0      tcp -- *      *       0.0.0.0/0     0.0.0.0/0
tcp dpt:80 state NEW recent: SET name: web side: source
0      0 ACCEPT    all -f *      *       0.0.0.0/0     0.0.0.0/0
limit: avg 100/sec burst 100
999 43956 ACCEPT    tcp -- *      *       0.0.0.0/0     0.0.0.0/0
tcp flags:0x16/0x02 limit: avg 20/sec burst 200
0      0 ACCEPT    all -- lo     *       0.0.0.0/0     0.0.0.0/0
204 14068 ACCEPT    all -- *      *       0.0.0.0/0     0.0.0.0/0
state RELATED,ESTABLISHED
1      44 ACCEPT    tcp -- *      *       0.0.0.0/0     0.0.0.0/0
multiport dports 80,443,25,465,110,995,143,993,587,465,22
0      0 LOG       all -- *      *       0.0.0.0/0     0.0.0.0/0
recent: UPDATE seconds: 60 hit_count: 20 name: web side: source LOG flags 0 level
4 prefix 'HTTP attack: '
0      0 DROP      all -- *      *       0.0.0.0/0     0.0.0.0/0
recent: UPDATE seconds: 60 hit_count: 20 name: web side: source
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target      prot opt in      out     source        destination
0      0 ACCEPT    icmp -- *      *       0.0.0.0/0     0.0.0.0/0
```

```

icmp type 8 limit: avg 1/sec burst 10
Chain OUTPUT (policy ACCEPT 17 packets, 1596 bytes)
pkts bytes target      prot opt in     out     source                   destination
0      0 ACCEPT all    --  *     lo     0.0.0.0/0                0.0.0.0/0

```

2) 在这台机器上开启 Nginx 或 Apache 服务, 即开启 80 端口; 在另一台 IP 为 192.168.1.211 的机器上运行 Webbench 压力测试工具, 来模拟非法攻击, 命令如下:

```

webbench -t 1000 -c 500 http://192.168.1.204
Webbench - Simple Web Benchmark 1.5
Copyright (c) Radim Kolar 1997-2004, GPL Open Source Software.

Benchmarking: GET http://192.168.1.204/
500 clients, running 1000 sec.

```

3) 观察结果可以得知, 在相当长的时间内 (即时间间隔为 1000s 的时间段内), IP 地址为 192.168.1.211 的机器是访问不了 192.168.1.204 的 Web 服务的, 只有过了这段压力测试的时间后, 192.168.1.211 的机器才能正常访问 192.168.1.204 的 Web 服务, 证明脚本生效; 还可以查看服务器的 iptables 日志, 输入以下命令即可查看系统的最后 100 条日志 (iptables 日志默认是放在 /var/log/messages 里的):

```
tail -n100 /var/log/messages
```

结果显示如下:

```

Oct 28 07:00:07 fabric kernel: HTTP attack: IN=eth0 OUT= MAC=00:16:3e:05:23:85:
00:16:3e:0a:10:7a:08:00 SRC=192.168.1.211 DST=192.168.1.204 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=56870 DF PROTO=TCP SPT=40874 DPT=80 WINDOW=14480 RES=0x00
SYN URGP=0
Oct 28 07:00:07 fabric kernel: HTTP attack: IN=eth0 OUT= MAC=00:16:3e:05:23:85:
00:16:3e:0a:10:7a:08:00 SRC=192.168.1.211 DST=192.168.1.204 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=48892 DF PROTO=TCP SPT=40875 DPT=80 WINDOW=14480 RES=0x00
SYN URGP=0
Oct 28 07:00:07 fabric kernel: HTTP attack: IN=eth0 OUT= MAC=00:16:3e:05:23:85:
00:16:3e:0a:10:7a:08:00 SRC=192.168.1.211 DST=192.168.1.204 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=201 DF PROTO=TCP SPT=40876 DPT=80 WINDOW=14480 RES=0x00
SYN URGP=0
Oct 28 07:00:07 fabric kernel: HTTP attack: IN=eth0 OUT= MAC=00:16:3e:05:23:85:
00:16:3e:0a:10:7a:08:00 SRC=192.168.1.211 DST=192.168.1.204 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=6778 DF PROTO=TCP SPT=40877 DPT=80 WINDOW=14480 RES=0x00
SYN URGP=0
Oct 28 07:00:07 fabric kernel: HTTP attack: IN=eth0 OUT= MAC=00:16:3e:05:23:85:
00:16:3e:0a:10:7a:08:00 SRC=192.168.1.211 DST=192.168.1.204 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=42 DF PROTO=TCP SPT=40878 DPT=80 WINDOW=14480 RES=0x00 SYN
URGP=0
Oct 28 07:00:07 fabric kernel: HTTP attack: IN=eth0 OUT= MAC=00:16:3e:05:23:85:
00:16:3e:0a:10:7a:08:00 SRC=192.168.1.211 DST=192.168.1.204 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=19679 DF PROTO=TCP SPT=40879 DPT=80 WINDOW=14480 RES=0x00
SYN URGP=0
Oct 28 07:00:07 fabric kernel: HTTP attack: IN=eth0 OUT= MAC=00:16:3e:05:23:85:

```

```

00:16:3e:0a:10:7a:08:00 SRC=192.168.1.211 DST=192.168.1.204 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=4841 DF PROTO=TCP SPT=40880 DPT=80 WINDOW=14480 RES=0x00
SYN URGP=0
Oct 28 07:00:07 fabric kernel: HTTP attack: IN=eth0 OUT= MAC=00:16:3e:05:23:85:
00:16:3e:0a:10:7a:08:00 SRC=192.168.1.211 DST=192.168.1.204 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=49662 DF PROTO=TCP SPT=40881 DPT=80 WINDOW=14480 RES=0x00
SYN URGP=0
Oct 28 07:00:07 fabric kernel: HTTP attack: IN=eth0 OUT= MAC=00:16:3e:05:23:85:
00:16:3e:0a:10:7a:08:00 SRC=192.168.1.211 DST=192.168.1.204 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=50257 DF PROTO=TCP SPT=40882 DPT=80 WINDOW=14480 RES=0x00
SYN URGP=0
[root@fabric ~]# tail /var/log/messages
Oct 28 07:00:07 fabric kernel: HTTP attack: IN=eth0 OUT= MAC=00:16:3e:05:23:85:
00:16:3e:0a:10:7a:08:00 SRC=192.168.1.211 DST=192.168.1.204 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=3805 DF PROTO=TCP SPT=40873 DPT=80 WINDOW=14480 RES=0x00
SYN URGP=0
Oct 28 07:00:07 fabric kernel: HTTP attack: IN=eth0 OUT= MAC=00:16:3e:05:23:85:
00:16:3e:0a:10:7a:08:00 SRC=192.168.1.211 DST=192.168.1.204 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=56870 DF PROTO=TCP SPT=40874 DPT=80 WINDOW=14480 RES=0x00
SYN URGP=0
Oct 28 07:00:07 fabric kernel: HTTP attack: IN=eth0 OUT= MAC=00:16:3e:05:23:85:
00:16:3e:0a:10:7a:08:00 SRC=192.168.1.211 DST=192.168.1.204 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=48892 DF PROTO=TCP SPT=40875 DPT=80 WINDOW=14480 RES=0x00
SYN URGP=0
Oct 28 07:00:07 fabric kernel: HTTP attack: IN=eth0 OUT= MAC=00:16:3e:05:23:85:
00:16:3e:0a:10:7a:08:00 SRC=192.168.1.211 DST=192.168.1.204 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=201 DF PROTO=TCP SPT=40876 DPT=80 WINDOW=14480 RES=0x00
SYN URGP=0
Oct 28 07:00:07 fabric kernel: HTTP attack: IN=eth0 OUT= MAC=00:16:3e:05:23:85:
00:16:3e:0a:10:7a:08:00 SRC=192.168.1.211 DST=192.168.1.204 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=6778 DF PROTO=TCP SPT=40877 DPT=80 WINDOW=14480 RES=0x00
SYN URGP=0
Oct 28 07:00:07 fabric kernel: HTTP attack: IN=eth0 OUT= MAC=00:16:3e:05:23:85:
00:16:3e:0a:10:7a:08:00 SRC=192.168.1.211 DST=192.168.1.204 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=42 DF PROTO=TCP SPT=40878 DPT=80 WINDOW=14480 RES=0x00 SYN
URGP=0
Oct 28 07:00:07 fabric kernel: HTTP attack: IN=eth0 OUT= MAC=00:16:3e:05:23:85:
00:16:3e:0a:10:7a:08:00 SRC=192.168.1.211 DST=192.168.1.204 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=19679 DF PROTO=TCP SPT=40879 DPT=80 WINDOW=14480 RES=0x00
SYN URGP=0
Oct 28 07:00:07 fabric kernel: HTTP attack: IN=eth0 OUT= MAC=00:16:3e:05:23:85:
00:16:3e:0a:10:7a:08:00 SRC=192.168.1.211 DST=192.168.1.204 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=4841 DF PROTO=TCP SPT=40880 DPT=80 WINDOW=14480 RES=0x00
SYN URGP=0
Oct 28 07:00:07 fabric kernel: HTTP attack: IN=eth0 OUT= MAC=00:16:3e:05:23:85:
00:16:3e:0a:10:7a:08:00 SRC=192.168.1.211 DST=192.168.1.204 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=49662 DF PROTO=TCP SPT=40881 DPT=80 WINDOW=14480 RES=0x00
SYN URGP=0
Oct 28 07:00:07 fabric kernel: HTTP attack: IN=eth0 OUT= MAC=00:16:3e:05:23:85:
00:16:3e:0a:10:7a:08:00 SRC=192.168.1.211 DST=192.168.1.204 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=50257 DF PROTO=TCP SPT=40882 DPT=80 WINDOW=14480 RES=0x00
SYN URGP=0

```


我们监测到大量带有 HTTP attack 日志头的 iptables 日志，如果发现此 IP 重复数量非常多，则直接用 iptables -I 命令将其禁止掉，省得它下次又重新发包攻击。

这里需要注意的是，iptables 的 recent 模块功能虽然强大，但它并不适用于一些基于 LVS+Keepalived 的 Linux 集群环境，这是因为后端的 Web 都是通过前端的 LVS 负载均衡器来连接的，有时会遇到并发数比较大的情况，比如单位时间内的并发数超过 2000，这时候用 recent 模块肯定是不行的，另外为了保证数据包转包的高效，在采用了 LVS+Keepalived 集群架构的机器上往往要关掉 iptables 防火墙。大家都知道，LVS/DR 模式是基于公网地址的，现在 SSH 暴力破解工具比比皆是，我们又应该如何防止 SSH 暴力破解呢？

6.9.4 利用 DenyHosts 工具和脚本来防止 SSH 暴力破解

笔者用 Nagios 外网监控服务器进行测试时设置的密码是 redhat_123456，可放进公网的第一天就被人采用暴力破解的手段更改了 root 密码，后来环境部署成熟以后发现仍然有不少外网 IP 在扫描和试探，看来不用点工具不行啊。于是想到了 DenyHosts，它是用 Python 2.3 写的一个程序，会分析 /var/log/secure 等日志文件，当发现同一 IP 在进行多次 SSH 密码尝试时就会将该 IP 记录到 /etc/hosts.deny 文件上，从而达到自动屏蔽该 IP 的目的。

DenyHosts 官方网站为：<http://denyhosts.sourceforge.net>。

安装 DenyHosts 的详细步骤如下。

1. 检查安装条件

1) 首先判断系统安装的 sshd 是否支持 TCP_Wrappers (默认都是支持的)，命令如下：

```
ldd /usr/sbin/sshd | grep libwrap.so.0
libwrap.so.0 => /lib64/libwrap.so.0 (0x00007f018d14d000)
```

2) 然后判断默认安装的 Python 版本，命令如下：

```
python -V
Python 2.6.6
```

CentOS 6.4 x86_64 已默认安装了 Python 2.6.6。

2. 安装及配置 DenyHosts 工具

在确认系统已安装 Python 2.3 以上版本的情况下，执行以下步骤。

1) 安装 DenyHosts，命令如下：

```
# cd /usr/local/src
# wget http://jaist.dl.sourceforge.net/sourceforge/denyhosts/DenyHosts-2.6.tar.gz
# tar xzf DenyHosts-2.6.tar.gz
# cd DenyHosts-2.6
# python setup.py install
```

程序脚本自动安装到 /usr/share/denyhosts。

库文件自动安装到 `/usr/lib/python2.3/site-packages/DenyHosts`。

`denyhosts.py` 自动安装到 `/usr/bin`。

2) 设置启动脚本, 命令如下所示:

```
# cd /usr/share/denyhosts/
# cp daemon-control-dist daemon-control
# chown root daemon-control
# chmod 700 daemon-control
# grep -v "^#" denyhosts.cfg-dist > denyhosts.cfg
# vim denyhosts.cfg
```

我们可以根据自己的需要对文件 `denyhosts.cfg` 进行相应的修改, 命令如下:

```
SECURE_LOG = /var/log/secure
```

上面表示 RedHat/CentOS 系统中安全日志的文件位置。

其他版本的 Linux 可根据 `denyhosts.cfg-dist` 内的提示进行选择, 命令如下:

```
PURGE_DENY = 30m
```

表示过多久后清除。

```
DENY_THRESHOLD_INVALID = 1
```

表示允许无效用户 (`/etc/passwd` 未列出) 登录失败的次数。

```
DENY_THRESHOLD_VALID = 5
```

表示允许有效 (普通) 用户登录失败的次数。

```
DENY_THRESHOLD_ROOT = 3
```

表示允许 root 登录失败的次数。

```
HOSTNAME_LOOKUP=NO
```

表示是否做域名反解。

如果需要 DenyHosts 随系统重启而自动启动, 还需要做如下设置, 先使用如下命令:

```
vim /etc/rc.local
```

然后加入下面这条命令:

```
/usr/share/denyhosts/daemon-control start
```

3) 启动 DenyHosts 工具, 命令如下:

```
/usr/share/denyhosts/daemon-control start
```

如果要使 DenyHosts 每次重启后都自动启动, 还需要做如下设置:

```
# cd /etc/init.d
ln -s /usr/share/denyhosts/daemon-control denyhosts
chkconfig --add denyhosts
```

```
chkconfig --level 345 denyhosts on
```

然后就可以启动了, 启动命令如下:

```
service denyhosts start
```

DenyHosts 配置文件的语法如下:

```
SECURE_LOG = /var/log/secure
```

这个就是 SSH 日志文件, 它是根据这个文件来判断非法 IP 的。

```
HOSTS_DENY = /etc/hosts.deny
```

这是控制用户登录的文件。

```
PURGE_DENY = 5m
```

表示要过多久后才能清除已经禁止的 IP, 这个可根据具体时间而定。

```
BLOCK_SERVICE = sshd
```

表示禁止的服务名。

```
DENY_THRESHOLD_INVALID = 1
```

表示允许无效用户失败的次数。

```
DENY_THRESHOLD_VALID = 10
```

表示允许普通用户登录失败的次数。

```
DENY_THRESHOLD_ROOT = 5
```

表示允许 root 登录失败的次数。

```
HOSTNAME_LOOKUP=NO
```

表示是否做域名反解。

```
DAEMON_LOG = /var/log/denyhosts
```

这是自己的日志文件。

```
ADMIN_EMAIL = yuhongchun027@163.com
```

这是管理员的邮件地址, 它会给管理员发邮件, CentOS 6.4 默认是开启了 Postfix 邮件服务的。

下面是全自动下载安装的小脚本 `install_denyhosts.sh` (以下脚本在 CentOS 5.5/5.6/6.4 x86_64 下均已测试通过), 当然安装后还得手动调整配置文件。脚本代码如下:

```
#!/bin/bash
cd /usr/local/src
wget http://jaist.dl.sourceforge.net/sourceforge/denyhosts/DenyHosts-2.6.tar.gz
tar xzf DenyHosts-2.6.tar.gz
cd DenyHosts-2.6
python setup.py install
```

```

cd /usr/share/denyhosts/
cp daemon-control-dist daemon-control
chown root daemon-control
chmod 700 daemon-control
grep -v "^#" denyhosts.cfg-dist > denyhosts.cfg
echo "/usr/share/denyhosts/daemon-control start" >>/etc/rc.local
cd /etc/init.d
ln -s /usr/share/denyhosts/daemon-control denyhosts
chkconfig --add denyhosts
chkconfig --level 345 denyhosts on
service denyhosts start

```

如果脚本成功运行，则会在最后一行有如下显示：

```

starting DenyHosts:    /usr/bin/env python /usr/bin/denyhosts.py --daemon
--config=/usr/share/denyhosts/denyhosts.cfg

```

还可以查看 DenyHosts 服务的状态，命令如下：

```
service denyhosts status
```

结果如下：

```
DenyHosts is running with pid = 9236
```

下面是 denyhosts 的示例，如果在 /etc/hosts.deny 里已有记录的 IP 机器仍然想连接安装了 DenyHosts 的机器，则会被拒绝，命令如下：

```
ssh 192.168.0.154
```

结果显示如下所示：

```

root@192.168.0.154's password:
Permission denied, please try again.
root@192.168.0.154's password:
Permission denied, please try again.
root@192.168.0.154's password:
Permission denied (publickey,gssapi-with-mic,password)

```

出现上面最后这行代码表示 /etc/hosts.deny 文件生效了，DenyHosts 工具部署成功。

DenyHosts 的原理很简单，其实就是收集 /var/log/secure 的信息，如果 root 登录失败的次数超过 10 次，则会将其写进 /etc/hosts.deny 文件里，从而达到禁止访问的目的。

通常，通过 SSH 登录远程服务器时，使用密码认证，分别输入用户名和密码，若两者满足一定的规则就可以登录。但是密码认证有以下缺点：

- ❑ 密码配置过长容易遗忘密码，比如笔者公司内部的开发服务器时不时就有进入单用户（Single User）改 root 密码的需求。
- ❑ 简单密码容易被人采用暴力手段破解。
- ❑ 服务器上的一个账户若要给多人使用，则必须让所有使用者都知道该账户的密码，导致密码容易泄露，而且如果有系统管理员要离职，修改密码时必须通知所有人。

而使用公钥认证则可以解决上述问题，其优点如下：

- ❑ 公钥认证允许使用空密码，省去了每次登录都需要输入密码的麻烦。
- ❑ 多个使用者可以通过各自的密钥登录到系统上的同一个账户。
- ❑ 用户的私匙还可以加密，安全系数比较高。
- ❑ 方便自动化运维部署。

综上所述，建议大家采用 SSH Key 认证登录的方式来取代传统的密码验证的登录方式。

6.10 TCP_Wrappers 应用级防火墙的介绍和应用

了解了 iptables 防火墙后，相信大家可能会对它强大的 IP 过滤功能感到惊叹。但是在日常工作中，有时仅仅过滤 IP 是满足不了我们的工作需求的，因此，这里再介绍一种基于应用级别的防火墙，那就是强大的 TCP_Wrappers。

为什么我们总说 Linux 服务器安全呢？通过下面的流程可以看出，一个客户端想要访问 Linux 服务器的资源，其实也不是一件容易的事情，它需要突破层层封锁和权限控制。

Linux 系统访问控制的流程如下：

客户端 → iptables → TCP_Wrappers → 服务本身的访问控制

具体说明如下：

- ❑ iptables：基于原 IP、目的 IP、原端口、目的端口来进行控制。
- ❑ TCP_Wrappers：对服务的本身进行控制。
- ❑ Service：对行为进行控制（它会结合文件和目录权限做更细致的控制）。

TCP_Wrappers 是根据 /etc/hosts.allow 及 /etc/hosts.deny 这两个文件来判断用户是否能够访问服务器资源的。其实，/etc/hosts.allow 与 /etc/hosts.deny 是 /usr/sbin/tcpd 的设定档，/usr/bin/tcpd 则是用来分析进入系统的 TCP 封包的一个软件，它是由 TCP Wrappers 提供的。那为什么叫作 TCP_Wrappers 呢？其中 Wrappers 有包裹的意思，也就是说，这个软件本身的功能就是分析 TCP 网络封包资料的！前面提到网络的封包资料是以 TCP 封包为主的，这个 TCP 封包的档头至少记录了来源与目标主机的 IP 与端口号，因此，通过分析 TCP 封包，就可以判断是否让这个客户端访问服务器资源。

TCP_Wrappers 的访问控制主要是通过以下两个文件来实现的：

```
/etc/hosts.allow
/etc/hosts.deny?
```

/etc/hosts.allow 用来定义允许的访问，/etc/hosts.deny 用来定义拒绝的访问。



注意 其实，在 /etc/hosts.allow 里是可以定义拒绝的访问的，/etc/hosts.deny 也可以做相反的工作，但不推荐大家这样操作，为什么我们要将简单的问题复杂化呢？

现在来了解一下 TCP_Wrappers 的访问控制判断顺序, 如果客户端 IP 通过了 iptables 防火墙后想访问我们的服务器资源, 这时系统会查看此客户端请求 IP 是否存在于 /etc/hosts.allow 列表里, 如果存在, 则接受此 IP 请求; 如果不存在, 则继续比对 /etc/hosts.deny 列表, 如果存在于 hosts.deny 列表里, 则拒绝此 IP 请求; 如果此 IP 在两个文件里都不存在, 则接受此 IP 请求, 其工作流程如图 6-11 所示。此流程图逻辑清晰, 建议大家也以此流程图进行记忆。

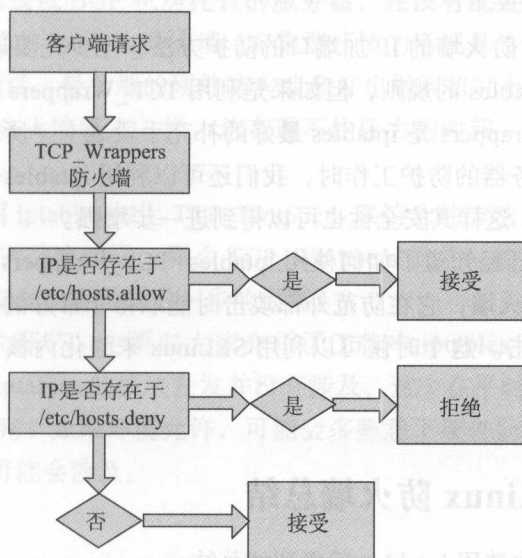


图 6-11 TCP_Wrappers 工作流程图

TCP_Wrappers 的语法格式如下:

```
<service>:<IP,domain,hostname...>:<allow|deny>
```

其语法很简单, 前面接服务名, 中间是“:”, 后面是 IP 或 IP 段, 最后面的 allow 或 deny 可以省略。这里跟大家说一个小知识: 在 Linux 系统里, 点分十进制是除了 iptables 外, 支持所有服务的语法, 比如我们可以 /etc/hosts.allow 里写上如下内容:

```
sshd:192.168.1.0/255.255.255.0
```

表示此服务器接受来自 192.168.1.0 到 255.255.255.0 网段所有机器的 SSH 请求。下面请大家思考一个比较复杂的问题, 如果一个 IP 既存在于 /etc/hosts.allow 里, 又存在于 /etc/hosts.deny 里 (例如 192.168.1.102 这个 IP 既存在于服务器 192.168.1.104 的 /etc/hosts.allow 里, 也存在于 /etc/hosts.deny 里), 那么这个 IP 会被服务器接受吗? 其实用上面的流程图一判断, 就可以得出结论, 192.168.1.104 是允许 192.168.1.102 的机器 SSH 登录的。

那么, 我们该如何利用 TCP_Wrappers 呢 (测试机器为 CentOS 5.8 x86_64, CentOS 6.4 x86_64 下面的 portmap 更改为 rpcbind, 这一点请大家注意区别)?

一些比较复杂的服务 (例如 NFS 服务) 在与客户端通信时会打开几个端口, 如果不

在配置文件里做修改，那么它通信的端口会与 vsftpd 服务的被动模式一样，也就是说端口都是随机的，这样我们用 iptables 来做安全策略时会非常麻烦，不过，如果通过 TCP_Wrappers 来设置相应的限制策略就会非常容易，例如通过配置 /etc/hosts.deny 文件来限制 192.168.1.102 的机器来访问本机的 NFS 资源，可以按如下方式编辑 /etc/hosts.deny 文件：

```
vim /etc/hosts.deny
portmap:192.168.1.102
```

如果要使用 iptables 防火墙的 IP 加端口的防护方法，至少先要通过抓包弄清楚 NFS 的端口问题才能够写出 iptables 的规则，但如果是利用 TCP_Wrappers 则可以很轻松地做好这个工作，可以说 TCP_Wrappers 是 iptables 最好的补充手段之一，所以很多朋友也称其为应用级防火墙。在做好服务器的防护工作时，我们还可以利用 iptables+TCP_Wrappers 的方法对服务器资源进行保护，这样其安全性也可以得到进一步增强。

到目前为止，我们已经知道了如何使用 iptables+TCP_Wrappers 建造及维护一个 Linux 系统 IP 级及应用级的防火墙，它在防范外部攻击时能取得非常好的效果，但是它们防范不了来自防火墙内部的攻击。这个时候可以利用 SELinux 来强化内核，使我们的计算机更加安全。

6.11 工作中的 Linux 防火墙总结

以下是笔者在工作中使用 iptables 后得到的总结。

1) iptables 防火墙并不能阻止 DDos 攻击，建议在项目实施中采购硬件防火墙，并将其置于整个系统之前，用于防止 DDos 攻击和端口映射；如果对安全有特殊要求，可再加上应用层级的防火墙，比如天泰应用防火墙，其功能会很强大。应用层级的防火墙能够基于对数据报文头部和载荷的完整检测，对 Web 应用客户端输入进行验证，从而对各类已知的及新兴的 Web 应用威胁提供全方位的防护，如 SQL 注入、跨站脚本、蠕虫、黑客扫描和攻击等。

2) 如果线上的机器采用了集群架构方案的话，建议关闭 iptables 防火墙，这样的目的是可以更好地提高后端服务器的网络性能，方便数据流在整个业务系统内部进行流通，安全方面的工作由硬件防火墙来承担。

3) 线上的机器（包括 AWS EC2 云主机）一般都有公网和内网两个 IP 地址，除了 Web 对外的业务（即 22、80 和 443）对公网开放以外，其他的尽量不要对公网开放，业务尽量走内网地址，比如 MySQL 或 Redis 业务等。

4) iptables 的 L 是命令，而 -v 和 -n 只是选项，它们不能进行组合，如 -Lvn；如果要列出防火墙的详细规则，可采用 iptables -nv -L 或 iptables -n -v -L。

5) 如果是使用远程来调试 iptables 防火墙，最好设置 Crontab 作业定时停止防火墙，以防止自己被锁定了，5 分钟停止一次 iptables 即可，等整个脚本完全稳定后再关闭此 Crontab 作业。

6) 如果使用默认禁止一切策略,在写防火墙策略时应该开放回环接口 lo (因为禁止一切也就包括了 lo 回环口),回环接口 lo 在 Linux 系统中被用来作为提供本地、基于网络服务的专用网络接口,不用通过网络接口驱动器来发送本地数据流,而是采用操作系统通过回环接口来发送,这大大地提高了性能;而关闭 lo 也会带来一些莫名其妙的问题,所以 iptables 脚本建议开启 lo 回环接口。

7) 如果是电信、双线或 BGP 机房托管的服务器,在没有配置前端硬件防火墙的情况下,建议开启 Linux 机器的 iptables 防火墙 (有集群环境的话视具体情况而定)。

8) 如果经费足够的话,最前端的硬件防火墙最好也做双机冗余,防止单防火墙出问题而导致整个网站崩溃,防火墙跟人一样,总有顶不住压力的时候,如果有双机的话,网站出问题的概率会小很多。

9) 工作中可以利用 iptables 加上 TCP_Wrappers 双结合的方法来对主机进行防护,如果说 iptables 是基于 IP 来过滤的话,那么 TCP_Wrappers 就是一种应用级的防火墙,两者结合起来 Linux 系统的安全性会得到进一步的提高。

上面的 iptables 相关环节,主要向大家介绍了工作中 iptables 经常会涉及的部分,而 iptables 的 mangle 链和 iptables 的模块开发并没有涉及,这个在平时的工作中用得很少,有兴趣的朋友可以自行研究。如果环境允许,可能会多熟悉下硬件防火墙的性能和特点,这些在我们的工作中很有可能会涉及。

6.12 Linux 服务器基础防护知识

现在许多生产服务器都是放置在 IDC 机房里的,有的并没有专业的硬件防火墙保护,我们应该如何做好其基础的安全措施呢?个人觉得应该从如下几个方面着手。

- 首先要保证自己的 Linux 服务器的密码绝对安全,笔者一般将 root 密码设置在 28 位以上,而且某些重要的服务器必须只有几个人知道 root 密码,这将根据公司管理层的权限来设置,如果有系统管理员离职,root 密码一定要更改。现在我们的做法一般是禁止 root 远程登录,只分配一个具有 sudo 权限的用户。服务器的账号管理一定要严格,服务器上除了 root 账号外,系统用户越少越好,如果非要添加用户来作为应用程序的执行者,请将他的登录 Shell 设为 nologin,即此用户是没有权利登录服务器的。终止未授权用户,定期检查系统有无多余的用户都是很有必要的工作。另外,像 vsftpd、samba 及 MySQL 的账号也要严格控制,尽可能只分配给他们满足基本工作需求的权限,而像 MySQL 等的账号,不要给任何用户 grant 权限。
- 防止 SSH 暴力破解是一个老生常谈的问题,解决这个问题有许多种方法:有的朋友喜欢用 iptables 的 recent 模块来限制单位时间内 SSH 的连接数,有的用 DenyHosts 防 SSH 暴力破解工具,应尽可能采用部署服务器密钥登录的方式,这样就算对外开放 SSH 端口,暴力破解也完全没有用武之地。

- ❑ 分析系统的日志文件，寻找入侵者曾经试图入侵系统的蛛丝马迹。last 命令是另外一个可以用来查找非授权用户登录事件的工具。last 命令输入的信息来自于 /var/log/wtmp。这个文件详细地记录着每个系统用户的访问活动。但是有经验的入侵者往往会删掉 /var/log/wtmp 以清除自己非法行为的证据，但是这种清除行为还是会露出蛛丝马迹：在日志文件里留下一个没有退出的操作和与之对应的登录操作（虽然在删除 wtmp 的时候登录记录没有了，但是待其登出的时候，系统还是会把它记录下来），不过高明入侵者会用 at 或 Crontab 等自己登出之后再删文件。
- ❑ 建议不定期用 `grep error /var/log/messages` 检查自己的服务器是否存在硬件损坏的情况，由于服务器长年搁置在机房中，最容易损坏的就是硬盘和风扇，因此在进行这些方面的日常维护时要格外注意，最好是定期巡视我们的 IDC 托管机房。
- ❑ 建议不定期使用 Chkrootkit 应用程序对 rootkit 的踪迹和特征进行查找，从它的报告中我们可以分析服务器否已经感染木马。
- ❑ 停掉一些系统不必要的服务，强化内核。多关注一下服务器的内核漏洞，现在 Linux 的很多攻击都是针对内核的，应尽量保证内核版本是最新的。

6.13 Linux 服务器高级防护知识

另外，我们还可以设计代码级别的 WAF 软件防火墙，主要是由 Nginx 结合 ngx_lua 模块来实现的，由于 Lua 语言的性能是接近于 C 语言的，而且 ngx_lua 模块本身就是基于为 Nginx 开发的高性能的模块，所以性能比较好。其可以实现的安全防护功能如下：

IP 黑白名单。

URL 黑白名单。

UserAgent 黑白名单。

Referer 黑白名单。

常见的 Web 漏洞防护，如 XSS、SQL 注入等。

CC 攻击防护。

扫描器简单防护。

其他业务安全需要的功能。

大家可以看到，其实代码级别的 WAF 防火墙比商业的 WAF 防火墙功能更加强大和灵活。

6.14 如何防止入侵

事实上，部分公司许多非核心的服务器并未放置于自己的机房内，并且未必有硬件防火墙保护，这个时候我们应该如何防止黑客入侵呢？本节将给出一些保证系统安全的建议，但这些建议并未涵盖全部的保证系统安全的方法，仅仅是几点建议。

- ❑ 系统的软件应尽可能地及时更新，特别是有重大安全隐患的软件。虽然经常更新计算机系统是本节中提到的保证系统安全的方法中最容易实现的，但是这项工作却经常被忽视。计算机最容易被攻破的情况是让其运行但却不对其进行更新。Linux 系统及开源软件最强的性能之一就是它们的高安全性，而这是有原因的。当一个安全问题被揭露时，开源社区会在很短的时间内提出解决方案，这为保证开源软件的高安全性提供了条件。在问题发现的同一天就找到修复方案也是很常见的，甚至是以以前从没有见过的安全问题也有很多都在发现的当天就被解决了。勤于更新软件是保证系统安全很重要的一方面，虽然推荐尽可能经常地更新软件，但是我们也要密切注意关注正在更新的软件，要保证所有的更新都不会影响正常的系统运行。
- ❑ 保证内部网的安全。很多时候为了安全，我们将核心生产服务器放置在公司内部的机房中，然后将注意力和精力放在外网的防护工作上面，于是疏忽了内部的安全性，这个时候服务器会很容易被人从内部进行破坏。正确的做法有许多种，比如应该将重要的生产服务器放在 DMZ 区域，跟我们的内网隔离开来，这样即使内部网络被人破坏或被人入侵，也不会影响生产服务器。像我们的线上环境，除了跳板机以外均没有开放公网 SSH 端口，而且重要区域的跳板机，只能允许特定的公网 IP 地址进行 SSH 连接。
- ❑ 应尽可能最小化安装服务器和运行最少的服务。通过这么多年的系统相关工作实践，我们发现，安装包最小的服务器相对而言是最为稳定的。而只提供必要的基础的核心服务，也是提高我们的服务器安全稳定性的方法之一。
- ❑ 我们在内核的强化上面也要做一些工作。多关注一下服务器的内核漏洞，毕竟现在有关 Linux 的很多攻击都是针对内核的，应尽可能地采用稳定的新内核版本，笔者公司现在用的内核版本为 2.6.32-358.el6.x86_64 和 3.14.35-28.38.amzn1.x86_64，分别对应的是 CentOS 系统和 AWS EC2 云主机系统。
- ❑ 如果条件允许的话，可以在我们的网站或系统关键位置的服务器上部署 snort，snort 是一个非常优秀的开源入侵检测软件，它集成了同类软件中最先进的技术，我们可以利用 snort 的警报找出攻击者而做出相应的防范措施。

6.15 小结

本章向大家详细介绍了 iptables 和 TCP_Wrappers 的语法，并演示了生产环境下的 iptables 安全脚本及其在 AWS EC2 主机中的应用。接着向大家介绍了一些比较实用的安全工具，比如命令行抓包工具 TCPDump、图形化抓包工具 Wireshark，以及安全扫描工具 Nmap 和 Hping，并且总结了 Linux 服务器在工作中可采取的安全措施，希望大家通过学习本章的内容能对系统安全防护概念有所了解，并掌握 Linux 服务器防护技巧，并且能够编写出适合自己服务器的 iptables 安全脚本。

Linux 集群及项目案例分享

作为一名资深系统管理员和系统架构设计师，笔者在工作中经常会遇到一些对外项目，比如中小型金融资讯网站和电子商务订单系统的架构及实施。在为客户实施项目方案时，客户基本上都会提出这样一条要求：保证服务的高可用性。基于此需求，我们所有的服务器，包括负载均衡器、文件服务器、Web 服务器、redis 缓存服务器，还有提供 MySQL 数据库的机器，基本上都有两台或两台以上的服务器。而且根据客户的要求及客户自身机房的硬件配置，我们还会选择不同的负载均衡器方案，比如硬件方面有 F5 和 Citrix NetScaler，软件方面有 LVS、Nginx、HAProxy 及 DNS 轮询，云计算服务产品有 Elastic Load Balancing。可以说在相当长的一段时间内，我的工作之一就是不停地测试它们，不停地完善和优化整体网站的架构设计。

在与一些系统管理员进行线下交流活动时，我发现不少技术很好的系统管理员由于公司自身环境等因素对 Linux 集群、负载均衡高可用等相关知识知之甚少，如果是从事 IT 的其他专业，对这方面的了解就更可想而知了。在这里，笔者希望通过分享自己的 Linux 集群项目经验，向大家说明什么叫负载均衡，什么叫高可用，什么叫 Linux 集群。与大家交流一下与之相关的专业知识，让大家走出误区，从真正意义上来理解它们。

7.1 负载均衡高可用核心概念及常用软件

7.1.1 什么是负载均衡高可用

在解释这个专业术语之前，我们需要先弄明白一个小问题，为什么需要负载均衡 (Load Balancing)？这里以一个示例来说明，假如我们有一个金融资讯类的网站，只允许 100 个用户同时在线访问。网站上线初期，由于知名度较小，加上没有宣传，只有几个用

户经常上线；后期知名度上去了，宣传也上去了，百度和谷歌也收录了我们的网站，这时，同时在线的用户数量直线上升，甚至会达到上千人；于是，网站变得异常繁忙，经常会反应不过来，这个时候用户势必会埋怨，为了不影响客户对我们的信心，一定要想办法解决这个问题。试想，如果有几台或几十台相同配置的机器，前端放一个转发器，轮流转发客户对网站的请求，每台机器都将用户数控制在 100 之内，那么网站的反应速度就会大大增快；即使其中的某台服务器因为硬件故障宕机了，也不会影响用户的访问。其中，这个神奇的转发器就是负载均衡器，英文名叫 Director。那么什么是负载均衡呢？负载均衡建立在现有的网络结构之上，它提供了一种廉价、有效、透明的方法来扩展网络设备和服务器的带宽，增加了吞吐量，加强了网络数据处理能力，提高了网络的灵活性和可用性？我们通过负载均衡器，可以实现 N 台廉价的 PC 服务器并行处理，使其计算能力达到小型机或大型机的水平，这也是目前负载均衡如此流行的主要原因。

高可用（High Availability, HA）其实有两种不同的含义，从广义上说，是指整个系统的高可用（High Availability）性；从狭义上说，一般是指主机的冗余接管，如主机 HA。如无特殊说明，本书中的 HA 都是指广义的高可用性。广义的高可用性是指能够保证整个系统不会因为某一台主机崩溃或故障损坏而发生停止服务的现象；狭义的就是我们前面提到的主机的冗余接管，下面我们可以从最前端的负载均衡器谈起了。

单台负载均衡器位于网站的最前端，它起着对客户请求进行分流的作用，相当于整个网站或系统的入口，如果它不幸崩溃（Crash）了，整个网站也会挂掉，所以这个时候要求有一种方案，能在短时间（这个时间一般要求小于 1 秒）内将崩溃的负载均衡器接管过去，这就称为高可用。这个时间非常短，客户完全不会察觉到其中的一台机器已经发生了崩溃的情况。至于负载均衡器后端的 Web 集群、数据库集群，因为有负载均衡器的内部机制，即使是其中的某一台或两台发生问题，也不会影响整套系统的使用，这种意义上的高可用就是广义上的。

现在我们俗称的 Linux 集群（Cluster），指的就是大范围内的整套系统架构，相对于负载均衡器后端的 Web 集群、Resin 集群或 MySQL 集群来说，它的涵盖面要广得多，包括负载均衡和高可用。这里为了便于区别，在提到集群时一般会带上前缀，比方说 Web 集群，指的就是后端提供相同服务的 Web 机器群；如果是 Linux 集群，指的就是大范围的系统集群架构，希望大家不要混淆。

目前，线上环境中应用得比较多的负载均衡器硬件有 F5 BIG-IP 和 Citrix NetScaler，软件有 LVS、Nginx 及 HAProxy，高可用软件有 Heartbeat、Keepalived，成熟的 Linux 集群架构有 DNS 轮询、LVS+Keepalived、Nginx/HAProxy+Keepalived 及 DRBD+Heartbeat，建议大家还可以关注下 AWS 的 Elastic Load Balancing。

7.1.2 以 F5 BIG-IP 作为负载均衡器

以硬件作为负载均衡器的主要有 F5 BIG-IP 和 Citrix NetScaler，CDN 机房最常见的硬

件负载均衡设备就是这个大名鼎鼎的 F5 BIG-IP。

笔者之前在项目实施过程中用的主要也是 F5 BIG-IP，这里对其进行简略介绍。F5 BIG-IP 的官方名称为本地流量管理器，可以做 4~7 层的负载均衡，具有负载均衡、应用交换、会话交换、状态监控、智能网络地址转换、通用持续性、响应错误处理、IPv6 网关、高级路由、智能端口镜像、SSL 加速、智能 HTTP 压缩、TCP 优化、第 7 层速率整形、内容缓冲、内容转换、连接加速、高速缓存、Cookie 加密、选择性内容加密、应用攻击过滤、拒绝服务 (DoS) 攻击和 SYN 洪水攻击保护、包过滤防火墙、包消毒等功能。

以下是 F5 BIG-IP 用作 HTTP 负载均衡器的主要功能：

- ❑ 提供了 12 种灵活的算法将所有流量均衡地分配到各个服务器，而面对用户，它只是一台虚拟服务器。
- ❑ 用于确认应用程序能否针对请求返回相应的数据。假如 F5 BIG-IP 后面的某一台服务器发生服务停止、死机等故障，F5 会检查出来并将该服务器标识为宕机，从而避免将用户的访问请求传送到该台发生故障的服务器上。只要其他的服务器能正常工作，用户的访问就不会受到影响。宕机的服务器一旦修复，F5 BIG-IP 就会自动查证，在了解到其能对客户请求做出正确的响应时即恢复向该服务器传送请求。
- ❑ 具有动态 Session 的会话保持功能。



注意 现阶段由于成本的原因，且为了使可控性更灵活，不会再采用硬件负载均衡的方案了，基本上会选择免费的开源软件方案。

7.1.3 以 LVS 作为负载均衡器

LVS 全称为 Linux Virtual Server，是章文嵩博士（现淘宝网基础核心软件研发负责人）主持的自由项目。它是一个负载均衡 / 高可用性集群，主要针对大业务量的网络应用（如新闻服务、网上银行、电子商务等）。LVS 建立在一个主控服务器（通常为双机）及若干真实服务器（Real-Server）所组成的集群之上。真实服务器负责实际提供服务，主控服务器根据指定的调度算法对真实服务器进行控制。而集群的结构对于用户来说是透明的，客户端只与单个的 IP（集群系统的虚拟 IP）进行通信，也就是说从客户端的视角来看，这里只存在单个服务器。真实服务器可以提供众多服务，如 FTP、HTTP、DNS、Telnet、SMTP，现在比较流行的还有将其用于 MySQL 集群等。主控服务器负责对真实服务器进行控制。客户端在向 LVS 发出服务请求时，负载均衡器会通过特定的调度算法来指定由某个 Real-Server 来应答请求，而客户端只与负载均衡的 IP（即虚拟的 VIP）进行通信，LVS 技术现在已经是一种成熟的负载均衡技术了，更多详情请大家参考官方文档。

1. LVS 的新模式 FULLNAT 简介

在大规模的网络应用中，比如在淘宝的业务中，官方 LVS 的 3 种模式（即大家熟悉的

DR、NAT 及 TUNNEL 模式) 都满足不了需求, 原因有以下 3 点:

- 3 种转发模式的部署成本都比较高。
- 和商用的负载均衡相比, LVS 没有 DDoS 攻击防御功能。
- 主备部署模式使得性能无法扩展。

下面来展开描述一下第一点, 即 LVS 转发模式的部署成本高的缺点。

- DR 的不足: 必须要求 LVS 跟后端所有的 REPLY 放在同一个 VLAN 里。当然有人会提出来分几个区, 每个区布一个 LVS, 但一个区的 VM 资源没有了, 就只能用其他区的 VM, 而用户需要将这些 VM 挂到同一个 VIP 下, 这是无法实现的。
- NAT 的不足: NAT 最主要的问题就是配置处理很复杂。以前在购买商业设备的时候, 我们需要在交换机上配置策略路由, OUT 方向的策略路由; 因为, 我们在工作中会考虑部署多套负载均衡方案, 如果选择默认路由就只能实现一套负载均衡方案。
- TUNNEL 的不足: 隧道的问题主要也是配置较复杂, 真实服务器需要加载一个 IP 模块, 同时进行一些配置。

针对上述问题, 相应的解决方法就是采用新转发模式 FULLNAT, 实现真实服务器间跨 VLAN 通信, 并且 IN/OUT 流都经过 LVS。

针对第二点, 官方 LVS 缺少 DDoS 攻击防御模块的缺点, 解决方法是: 采用 SYNPROXY (SYNFlood 攻击防御模块) 和其他 TCP FLAG DDoS 攻击防御策略。

针对第三点, 性能无法线性扩展的缺点, 解决方法是: 采用 Cluster 部署模式。

LVS 在大规模网络环境中应用的更多详细资料请参考 <http://blog.aliyun.com/1750>。

2. 算法简介

选定了转发方式后, 采用哪种调度算法将决定整个负载均衡的性能表现。不同的算法适用于不同的应用场合, 有时可能需要针对特殊场合, 自行设计调度算法。每个负载均衡器都有自己独有的算法, 下面跟大家介绍下 LVS/HAProxy/Nginx 常见的算法。

首先是 LVS 的常见算法。

(1) 轮叫调度 (Round Robin)

调度器通过“轮叫”调度算法将外部请求按顺序轮流分配到集群中的真实服务器上, 它均等地对待每一台服务器, 而不管服务器上实际的连接数和系统负载。任何形式的负载均衡器 (包括硬件或软件级别的) 都带有基本的轮叫功能 (也叫轮询功能)。

(2) 加权轮叫 (Weighted Round Robin)

调度器通过“加权轮叫”调度算法根据真实服务器的不同处理能力来调度访问请求。这样就可以保证处理能力强的服务器能够处理更多的访问流量。调度器可以自动问询真实服务器的负载情况, 并动态地调整其权值。

(3) 最少链接 (Least Connections)

调度器通过“最少链接”调度算法动态地将网络请求调度到已建立的链接数最少的服

服务器上。如果集群系统的真实服务器具有相近的系统性能,采用“最少链接”调度算法可以较好地均衡负载。

(4) 加权最少链接 (Weighted Least Connections)

在集群系统中的服务器性能差异较大的情况下,调度器采用“加权最少链接”调度算法优化负载均衡的性能,具有较高权值的服务器将承受较大比例的活动连接负载。调度器可以自动问询真实服务器的负载情况,并动态地调整其权值。

(5) 基于局部性的最少链接 (Locality-Based Least Connections, LBLC)

“基于局部性的最少链接”调度算法是针对目标 IP 地址的负载均衡,目前主要用于 Cache 集群系统。该算法根据请求的目标 IP 地址找出该目标 IP 地址最近使用的服务器,若该服务器是可用的且没有超载,则将请求发送到该服务器上;若服务器不存在,或者该服务器超载且有服务器处于一半的工作负载,则用“最少链接”的原则选出一个可用的服务器,将请求发送到该服务器上。

(6) 带复制的基于局部性的最少链接 (Locality-Based Least Connections with Replication)

“带复制的基于局部性的最少链接”调度算法也是针对目标 IP 地址的负载均衡,目前主要用于 Cache 集群系统。它与 LBLC 算法的不同之处是它要维护的是从一个目标 IP 地址到一组服务器的映射,而 LBLC 算法维护的是从一个目标 IP 地址到一台服务器的映射。该算法根据请求的目标 IP 地址找出该目标 IP 地址对应的服务器组,按“最少链接”原则从服务器组中选出一台服务器,若服务器没有超载,则将请求发送到该服务器;若服务器超载,则按“最少链接”原则从这个集群中选出一台服务器,将该服务器加入到服务器组中,将请求发送到该服务器。同时,若该服务器组有一段时间没有被修改,则将最忙的服务器从服务器组中删除,以降低复制的程度。

(7) 目标地址散列 (Destination IP Hashing)

“目标地址散列”调度算法根据请求的目标 IP 地址,作为散列键 (Hash Key) 从静态分配的散列表中找出对应的服务器,若该服务器是可用的且未超载,则将请求发送到该服务器,否则返回空。

(8) 源地址散列 (Source IP Hashing)

“源地址散列”调度算法根据请求的源 IP 地址,作为散列键从静态分配的散列表中找出对应的服务器,若该服务器是可用的且未超载,则将请求发送到该服务器,否则返回空。

(9) 源 IP 端口散列 (Source IP Port Hashing)

通过 Hash 函数将来自同一个源 IP 地址和源端口号的请求映射到后端的同一台服务器上,该算法适用于需要保证来自同一用户同一业务的请求被分发到同一台服务器的场景。

(10) 随机 (Random)

随机地将请求分发到不同的服务器上,从统计学的角度来看,调度的结果为各台服务器平均分担用户的连接请求,该算法适用于集群中各机器性能相当而且无明显优劣差异的场景。

下面来看看 HAProxy 的常见算法。

HAProxy 的算法现在也越来越多了，具体有如下 8 种。

- ❑ **roundrobin**：表示简单的轮询，这个不用多说，负载均衡基本都具备这一算法。
- ❑ **static-rr**：每个服务器根据权重轮流使用，类似 roundrobin，但它是静态的，这意味着运行时修改权重是无效的。另外，它对服务器的数量没有限制。
- ❑ **leastconn**：连接数最少的服务器优先接收连接。leastconn 建议用于长会话服务，例如 LDAP、SQL、TSE 等，而不适合于短会话协议，如 HTTP。该算法是动态的，对于实例启动慢的服务器权重会在运行中动态调整。
- ❑ **source**：对请求源 IP 地址进行哈希，用可用服务器的权重总数除以哈希值，根据结果进行分配。只要服务器正常，同一个客户端 IP 地址总是访问同一个服务器。如果哈希的结果随可用服务器数量的变化而变化，那么客户端会定向到不同的服务器。该算法一般用于不能插入 Cookie 的 TCP 模式。它还可以用于在广域网上为拒绝使用会话 Cookie 的客户端提供最有效的粘连。该算法默认是静态的，所以运行时修改服务器的权重是无效的，但是算法会根据“hash-type”的变化进行调整。
- ❑ **URI**：表示根据请求的 URI 地址进行哈希，用可用服务器的权重总数除以哈希值，根据结果进行分配。只要服务器正常，同一个 URI 地址总是访问同一个服务器。一般用于代理提供缓存服务的机器，以最大限度地提高缓存的命中率。该算法只能用于 HTTP 后端。该算法一般用于后端节点机器是缓存服务器的场景，它默认是静态的，所以运行时修改服务器的权重是无效的，但是算法会根据“hash-type”的变化进行调整。
- ❑ **url_param**：表示根据 URL 的参数来调度，用于将同一个用户的信息都发送到同一个后端服务器。在 HTTP GET 请求的查询串中查找 <param> 中指定的 URL 参数，基本上可以锁定使用特制的 URL 到特定的负载均衡器节点的要求，该算法一般用于将同一个用户的信息发送到同一个后端服务器。该算法默认是静态的，所以运行时修改服务器的权重是无效的，但是算法会根据“hash-type”的变化进行调整。
- ❑ **hdr(name)**：在每个 HTTP 请求中查找 HTTP 头 <name>，然后算法根据 HTTP 请求头来锁定每一次 HTTP 请求。如果缺少头或头没有任何值，则用 roundrobin 代替。该算法默认是静态的，所以运行时修改服务器的权重是无效的，但是算法会根据“hash-type”的变化进行调整。
- ❑ **rdp-cookie(name)**：查询每个进来的 TCP 请求并哈希 RDP cookie<name>，该机制用于退化的持久模式，可以使同一个用户或同一个会话 ID 总是发送给同一台服务器。如果没有 Cookie，则使用 roundrobin 算法代替。该算法默认是静态的，所以运行时修改服务器的权重是无效的，但是算法会根据“hash-type”的变化进行调整。

最后是 Nginx 的常见算法。

(1) 轮询（默认）

每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器宕机，则会跳过

该服务器分配至下一个监控的服务器。并且它无须记录当前所有连接的状态，所以它是一种无状态调度。

(2) weight

在轮询的基础上加上权重，weight 和访问比率成正比，即用于表明后端服务器的性能好坏，这种情况特别适合后端服务器性能不一致的工作场景。

(3) ip_hash

每个请求按访问 IP 的哈希结果进行分配，当新的请求到达时，先将其客户端 IP 通过哈希算法进行计算得出一个值，随后的请求客户端 IP 的哈希值只要相同，就会被分配到同一台后端服务器上，该调度算法可以解决 Session 的问题，但有时会导致分配不均即无法保证负载均衡。

(4) fair (第三方)

按后端服务器的响应时间来分配请求，响应时间短的优先分配。

(5) url_hash (第三方)

按访问 URL 的哈希结果来分配请求，使每个 URL 定向到同一个后端服务器，后端服务器作为缓存时比较有效。

(6) Tengine 增加的一致性哈希算法

这种算法应该是借鉴了目前最流行的一致性哈希算法思路。它的具体做法是：将每个 Server 虚拟成 N 个节点，均匀分布到哈希环上，每次请求时，根据配置的参数计算出一个哈希值，在哈希环上查找离这个哈希值最近的虚拟节点，对应的 Server 作为该次请求的后端机器，这样做的好处是如果动态增加了机器，或者某台 Web 机器宕机，对整个集群的影响会最小。

了解这些算法的原理能够在特定的应用场合选择最适合的调度算法，从而尽可能地保持 Real Server 的最佳利用性。当然也可以自行开发算法，不过这已超出本文范围，请参考有关算法原理的资料。

7.1.4 以 Nginx 作为负载均衡器

Nginx 在作为负载均衡器的同时也是反向代理服务器，其配置语法相当简单，可以按轮询、ip_hash、url_hash、权重等多种方法对后端的服务器做负载均衡，同时还支持后端服务器的健康检查。另外，它相对于 LVS 来说比较有优势的一点是，由于它是基于第 7 层的负载均衡，是根据报头内的信息来执行负载均衡任务的，因此对网络的依赖性比较小，理论上只要 ping 得通就能够实现负载均衡。在国内，Nginx 不仅可以作为一款性能优异的负载均衡器，同时也是一款适用于高并发环境的 Web 应用软件，在新浪、金山、迅雷在线等大型网站都有其相关应用。Nginx 作为负载均衡器的优点如下：

- ❑ 配置文件非常简单，风格跟程序一样通俗易懂。
- ❑ 成本低廉。Nginx 为开源软件，可以免费使用。而购买 F5 BIG-IP、NetScaler 等硬

件负载均衡交换机则需要十多万甚至几十万人民币。

- ❑ 支持 Rewrite 重写规则：能够根据域名、URL 的不同，将 HTTP 请求分配到不同的后端服务器群组上。
- ❑ 有内置的健康检查功能：即使 Nginx Proxy 后端的某台 Web 服务器宕机了，也不会影响前端访问。
- ❑ 节省带宽：支持 GZIP 压缩，可以添加浏览器本地缓存的 Header。
- ❑ 稳定性高：用于反向代理，宕机的概率微乎其微。通过跟踪一些已上线的网站和系统，我们发现在高并发的情况下，Nginx 作为负载均衡器 / 反向代理宕机的次数几乎是零。

它的缺点是目前只支持 HTTP 和 MAIL 的负载均衡，不过我们可以取长补短，根据其支持的 Rewrite 重写规则和稳定性高的特点，将其应用于大型网站中间级别的负载均衡层（七层负载均衡）上，如图 7-1 所示。

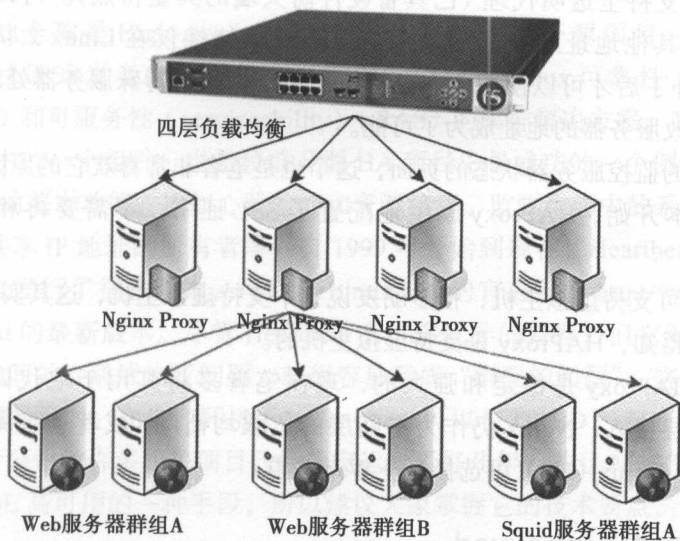


图 7-1 Nginx 作为中层负载均衡器的拓扑图

7.1.5 以 HAProxy 作为负载均衡器

HAProxy 是一款可提供高可用性、负载均衡，以及基于 TCP(第四层)和 HTTP(第七层)应用的代理软件。

- ❑ HAProxy 是完全免费的，借助 HAProxy 可以快速并可靠地提供基于 TCP 和 HTTP 应用的代理解决方案。HAProxy 最主要的特点是性能，HAProxy 特别适合于那些负载很大的 Web 站点，这些站点通常又需要会话保持或七层处理。HAProxy 完全可以支持数以万计的并发连接。并且 HAProxy 的运行模式可以使它既简单又安全

地整合到我们的网站系统架构中，同时还可以保护 Web 服务器不暴露到网络上（即通过防火墙 80 端口映射的方法）。HAProxy 也是一款优秀的负载均衡软件，其优点如下：

- ❑ 免费开源，稳定性也非常好，这点通过笔者做的一些项目就可以看出来，单 HAProxy 也表现得不错，其稳定性可以与硬件级的 F5 BIG-IP 相媲美。
- ❑ 根据官方文档可知，HAProxy 的单机带宽速度可以达到 10Gbit/s，这个数值作为软件级负载均衡器是相当惊人的，具体可以参考其官方说明 <http://haproxy.1wt.eu/10g.html>。
- ❑ HAProxy 支持连接拒绝。因为维护一个连接打开的开销是很低的，但有时我们必须限制攻击蠕虫（Attack Bots），也就是说通过限制它们的连接打开来防止它们的危害。这个功能已经拯救了很多被 DDoS 攻击的小型站点，这也是其他负载均衡器所不具备的。
- ❑ HAProxy 支持全透明代理（已具备硬件防火墙的典型特点）。可以用客户端 IP 地址或任何其他地址来连接后端服务器。这个特性仅在 Linux 2.4/2.6 内核修补了 cttproxy 补丁后才可以使使用，这个特性也使得为某些特殊服务器处理部分流量的同时又不修改服务器的地址成为了可能。
- ❑ 自带强大的监控服务器状态的页面，这个也是笔者非常喜欢它的原因之一。
- ❑ 从 1.5 版本开始，HAProxy 原生地配置了 SSL 证书，不需要再和 Stunnel 配合使用了。
- ❑ HAProxy 可支持虚拟主机，很多朋友说它不支持虚拟主机，这其实是个误区，通过测试可以得知，HAProxy 是支持虚拟主机的。

综上所述，HAProxy 是稳定和强大的，现在笔者多将其用于取代四层硬件防火墙作为网站的最外层接入，Nginx 仍作为中间层的负载均衡层（或者直接简化掉这层），即 HAProxy (LB) → Nginx (LB)(可选择) → Web 集群。

7.1.6 高可用软件 Keepalived

Keepalived 是一款优秀的、可实现高可用的软件，它运行在 LVS 之上，它的主要功能是实现真实机的故障隔离及负载均衡器间的失败切换（FailOver）。Keepalived 是一个类似于 Layer3、Layer4、Layer 5 交换机制的软件，也就是我们平时所说的第 3 层、第 4 层和第 5 层交换。Keepalived 的作用是检测 Web 服务器的状态，如果有一台 Web 服务器死机，或者工作出现故障，那么它将检测到该有故障的 Web 服务器，并将其从系统中剔除，待 Web 服务器工作正常后，Keepalived 又会自动将该 Web 服务器加入到服务器群中，这些工作全部自动完成，不需要人工干涉，需要人工做的只是修复出了故障的 Web 服务器。它的主要特点如下：

- ❑ Keepalived 是 LVS 的扩展项目，因此 LVS 和 Keepalived 之间具备良好的兼容

性。这点应该是 Keepalived 部署比其他类似工具更简洁的原因，尤其是相对于 Heartbeat 而言，Heartbeat 作为 HA 软件，其复杂的配置流程让许多新手望而生畏。

- 通过对服务器池对象的健康检查，实现对失效机器 / 服务器的故障隔离。
- 负载均衡器之间的失败切换（即 Failover）是通过 VRRPv2（Virtual Router Redundancy Protocol）stack 实现的，当初设计 VRRP 的目的就是解决静态路由器的单点故障问题。
- 通过实际的线上项目可以得知，iptables 的启用是不会影响 Keepalived 的运行的，但为了保证更好的性能，我们通常会将整套系统内所有主机的 iptables 都停用。
- Keepalived 产生的 VIP 就是我们整个系统对外的 IP，如果最外端的防火墙采用的是路由模式，那么我们就映射此内网 IP 为公网 IP。

Keepalived 是一款优秀的高可用软件，我们现在多将其应用于生产环境下的 LVS/HAProxy、Nginx 中，一般采取的都是双机方案，以保证网站最前端负载均衡器的高可用性。

7.1.7 高可用软件 Heartbeat

Linux-HA 的全称是 High-Availability Linux，它是一个开源项目。这个开源项目的目标是：通过社区开发者的共同努力，提供一个增强 Linux 可靠性（reliability）、可用性（availability）和可服务性（serviceability）（RAS）的集群解决方案。其中 Heartbeat 就是 Linux-HA 项目中的一个组件，也是目前开源 HA 项目中最成功的一个例子，它提供了所有 HA 软件所需要的基本功能，比如心跳检测和资源接管、监测集群中的系统服务、在集群中的节点间转移共享 IP 地址的所有者等。自 1999 年开始到现在，Heartbeat 在行业内得到了广泛的应用，也发行了很多版本，可以从 Linux-HA 的官方网站 <http://www.linux-ha.org> 上下载到 Heartbeat 的最新版本。尽管 Heartbeat 有许多优异的特性，但它配置起来非常麻烦，而且如果双机之间的心跳线出了问题，就很容易形成“脑裂的问题”，这也是目前制约其被大规模部署应用的原因。在生产环境下，Heartbeat 可以与 DRBD 一起应用于线上的高可用文件系统，笔者公司的许多相关项目已经稳定运行了好几年，并且 MySQL 官方也推荐将其作为实现 MySQL 高可用的一种手段，所以建议大家掌握它的技术要点，也可将其用于自己的网站或系统。

7.1.8 高可用块设备 DRBD

DRBD（Distributed Replicated Block Device）是一种块设备，可以用于高可用（HA）软件之中。它的功能类似于一个网络 RAID-1（工作原理见图 7-2）。当你将数据写入本地文件系统时，该数据还将被发送到网络中的另一台主机上，并以相同的形式记录在一个文件系统中。本地（主节点）与远程主机（备节点）的数据可以保证实时同步。当本地系统出现故障时，远程主机上还保留着一份相同的数据，可以继续使用。在高可用（HA）软件中使用 DRBD 功能，可以代替一个共享盘阵。因为数据同时存在于本地主机和远程主机上，切换时，远程主机只要使用它上面的那份备份数据就可以继续服务了。

DRBD 的工作原理如图 7-2 所示。

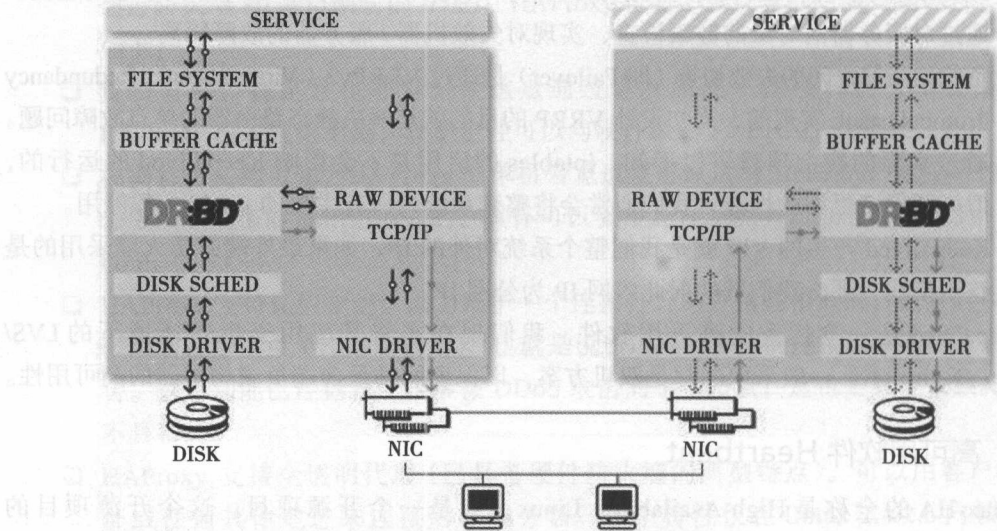


图 7-2 DRBD 工作原理图

DRBD 支持 3 种不同的复制模式，允许 3 种程度的复制同步。

- ❑ 协议 A：异步复制协议。只要主节点完成本地写操作就认为写操作完成，并且需要复制的数据包会被存放本地 TCP 发送缓存中。当发生 Failover 故障时，在 Standby 节点的数据仍被认为是稳固的，然而，在故障发生的时间点上很多最新的更新操作会丢失。
- ❑ 协议 B：内存同步（半同步，semi-synchronous）复制协议。只有当本地磁盘的写入已经完成，并且复制数据包已经到达对应从节点时，此时主节点才认为磁盘写入已经完成。通常情况下，发生 Failover 不会导致数据丢失（因为后备系统内存中已经获得了数据更新）。然而，如果所有节点同时出现电源故障，则主节点数据存储会发生不可逆的错误结构，主节点上多数最新写入的数据可能会丢失。
- ❑ 协议 C：同步复制协议。只有在本地和远程磁盘都确定写入已完成时，主节点才会认为写入完成。这样可确保发生单点故障时不会导致任何数据丢失。如果发生数据丢失的现象，那也只会所有节点同时存在错误存储时才会发生这种情况。

在 DRBD 设置中，最常用的复制协议是协议 C。选择哪种复制协议将受部署的两个因素影响：保护要求和延迟。为了保证数据的一致性和可靠性，建议选择协议 C。

另外，笔者公司在线上环境中主要采用 DRBD+Heartbeat+NFS 组成高可用的文件系统，此项目上线几年多一直没有发生过丢失数据的现象；另外，DRBD 已被 MySQL 官方采用并作为其推荐的高可用方案之一。

7.1.9 四、七层负载均衡工作流程对比

按照七层网络协议栈的层划分，负载均衡设备可以划分为四层负载均衡和七层负载均衡。其中，四层负载均衡是基于 IP+ 端口的，它能够对报文按 IP 进行分发，七层负载均衡是基于 URL 地址的服务器负载均衡，它能够针对七层报文内容进行解析，并根据其中的 URL 关键字进行逐包转发，比较常见的功能就是我们所说的“动静分离”（即将静态内容，如 JPG、HTML、CSS 和 JS 文件分发到 Nginx 服务器处理，PHP 或 JSP 动态文件分发到 Apache 服务器或 Tomcat 服务器处理）。四层负载均衡的典型代表是 LVS，七层负载均衡的典型代表是 Nginx 和 HAProxy（另注：HAProxy 既可以做四层均衡设备，又可以做七层负载均衡设备）。

下面以常见的 LVS/DR 模式来举例说明四层负载均衡的工作流程，如图 7-3 所示。

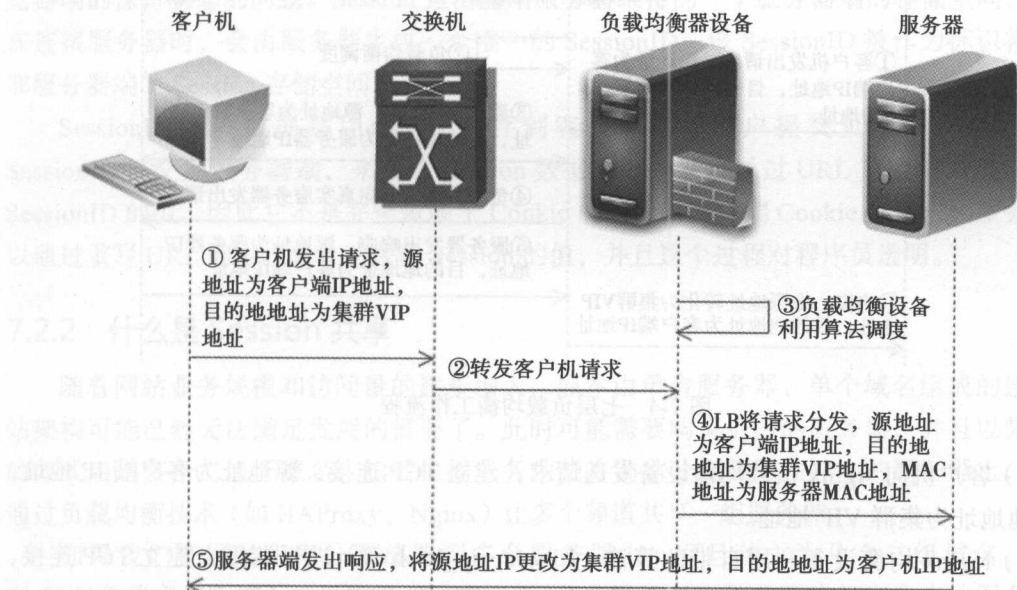


图 7-3 LVS/DR 四层负载均衡工作流程

如图 7-3 所示，LVS/DR 四层负载均衡工作流程如下：

- 1) 客户机向负载均衡设备发出请求，源地址为客户机的 IP 地址，目的地地址为整个集群的 VIP 地址。
- 2) 交换机转发客户机的请求。
- 3) LVS 负载均衡服务器利用自带的算法（一般是 wrr 或 wlc）进行算法调度，将请求转到后端的某一台真实的 Web 服务器。
- 4) 此时请求报文的源地址仍为客户机的 IP 地址，目的地地址为集群 VIP 地址，但 MAC 地址被 LVS 负载均衡服务器更改为后端的真实服务器 MAC 地址。

5) 后端的真实服务器发出响应，源地址为集群 VIP 地址，目的地地址为客户端 IP 地址，不通过 LVS 负载均衡服务器（报文仍然要经过交换机）直接与客户机发生联系，回应客户机最初发出的 HTTP 请求。

下面以常用的 Nginx 负载均衡设备来举例说明七层负载均衡的工作流程，如图 7-4 所示。如图 7-4 所示，七层负载均衡工作流程如下：

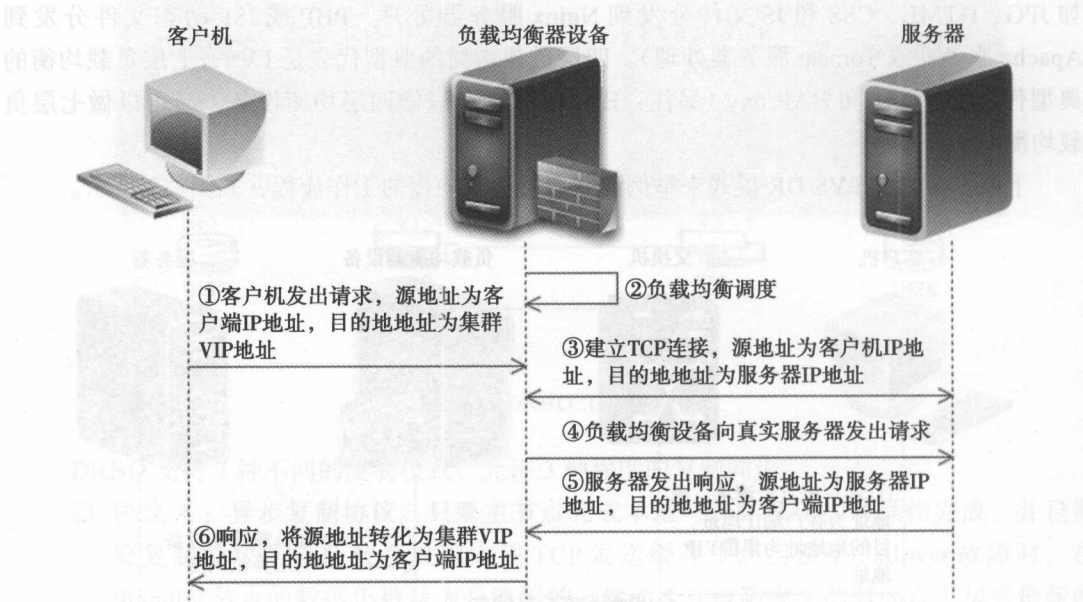


图 7-4 七层负载均衡工作流程

- 1) 客户机向 Nginx 负载均衡设备发送请求，建立 TCP 连接，源地址为客户端 IP 地址，目的地地址为集群 VIP 地址。
- 2) Nginx 均衡设备利用自带的算法（如 wrr、ip_hash 等）进行调度，建立 TCP 连接，将客户机的请求发送到后面的某一台真实的 Web 服务器上面，此时源地址为客户机 IP 地址，目的地地址为某台真实服务器的 IP 地址。
- 3) Nginx 负载均衡设备向后端的某台真实服务器发出请求。
- 4) 真实 Web 服务器端发出响应，此时源地址为真实服务器 IP 地址，目的地地址为客户端 IP 地址。
- 5) 报文经过 Nginx 七层负载均衡设备时，源地址被还原为集群 VIP 地址，目的地地址为客户端 IP 地址。

以上就是四七层负载均衡设备的工作流程，我们通过对比可以发现：四层负载均衡设备（如 LVS/DR）的优势在于面对大流量的冲击时，报文只是单方面经过四层负载均衡设备，负载均衡设备负担很小，不易成为网站或系统的瓶颈；而七层负载均衡在分流过程中

能够对应应用层协议进行深度识别，带来了更精细化均衡的可能，再加上 HTTP 协议应用广泛并且相对简单，所以七层负载均衡对 HTTP 请求进行负载均衡的商用能力最强，而四层负载均衡（LVS）因无法对七层业务实现按内容转发，限制了其适用范围，因此七层负载均衡（HAProxy 或 Nginx）目前已逐渐成为负载均衡技术的主流。

7.2 负载均衡关键技术

7.2.1 什么是 Session

Session 在网络应用中被称为“会话”，借助它可提供服务器端与客户端系统之间必要的交互。因为 HTTP 协议本身是无状态的，所以经常需要通过 Session 来解决服务器端和浏览器端的保持状态的问题。Session 是由应用服务器维持的一个服务器端的存储空间，用户在连接服务器时，会由服务器生成一个唯一的 SessionID，该 SessionID 被作为标识符来存取服务器端的 Session 存储空间。

SessionID 这一数据是用 Cookie 保存到客户端的，用户提交页面时，会将这一 SessionID 提交到服务器端，来存取 Session 数据。服务器也通过 URL 重写的方式来传递 SessionID 的值，因此它不是完全依赖于 Cookie 的。如果客户端 Cookie 禁用，则服务器可以通过重写 URL 的方式来自动保存 Session 的值，并且这个过程对程序员透明。

7.2.2 什么是 Session 共享

随着网站业务规模和访问量的逐步增大，原本由单台服务器、单个域名组成的迷你网站架构可能已经无法满足发展的需要了。此时可能需要购买更多的服务器，并且以频道化的方式启用多个二级子域名，然后根据业务功能将网站分别部署在独立的服务器上，或者通过负载均衡技术（如 HAProxy、Nginx）让多个频道共享一组服务器。

如果我们把网站程序分别部署到多台服务器上，而且独立为几个二级域名，由于 Session 存在实现原理上的局限性（PHP 中 Session 默认以文件的形式保存在本地服务器的硬盘上），因此使得网站用户不得不在几个频道间来回输入用户名和密码登入，导致用户体验大打折扣；另外，原本程序可以从用户 Session 变量中直接读取的资料（如：昵称、积分、登入时间等），因为无法跨服务器同步更新 Session 变量，迫使开发人员必须实时读写数据库，从而增加了数据库的负担。于是，解决网站跨服务器的 Session 共享问题的需求变得迫切起来，最终催生了多种解决方案，下面将列举 4 种较为可行的方案来进行对比和探讨。

1. 基于 Cookie 的 Session 共享

这个方案部分读者可能会觉得比较陌生，但它在大型网站中已被普遍应用了。其原理是将全站用户的 Session 信息加密、序列化后以 Cookie 的方式统一种植在根域名下（如：.host.com）。当浏览器访问该根域名下的所有二级域名站点时，将与域名相对应的所有

Cookie 内容的特性都传递给它，从而实现用户的 Cookie 化 Session 在多服务器间的共享访问。

该方案的优点是无需额外的服务器资源；缺点是由于受 HTTP 协议头信息长度的限制，仅能够存储小部分用户的信息，同时 Cookie 化的 Session 内容需要进行安全加解密（如：采用 DES、RSA 等进行明文加解密；再由 MD5、SHA-1 等算法进行防伪认证），另外它也会占用一定的带宽资源，因为浏览器会在请求当前域名下的任何资源时将本地 Cookie 附加在 HTTP 头中传递到服务器上。

2. 基于数据库的 Session 共享

首选当然是大名鼎鼎的 MySQL 数据库，并且建议使用内存表 Heap，以提高 Session 操作的读写效率。这个方案的实用性比较强，已被普遍使用，它的缺点在于 Session 的并发读写能力取决于 MySQL 数据库的性能，同时需要我们自己来实现 Session 淘汰逻辑，以便定时地从数据表中更新、删除 Session 记录，当并发过高时容易出现表锁，虽然我们可以选择行级锁的表引擎，但不得不承认使用数据库存储 Session 还是有些杀鸡用牛刀的架势。

3. Session 复制

熟悉 Tomcat 或 Weblogic 的朋友对 Session 复制应该是非常熟悉和了解了，它从字面意义上也非常好了解。Session 复制，就是将用户的 Session 复制到 Web 集群内的所有服务器上，Tomcat 或 Weblogic 自身都带有这种处理机制。但缺点也很明显：随着机器数量的增加，网络负担将成指数级上升，性能随着服务器数量的增加而急剧下降，而且很容易引起网络风暴。

4. 基于 Memcache/redis 的 Session 共享

Memcache 是一款基于 Libevent 的多路异步 I/O 技术的内存共享系统，简单的 Key + Value 数据存储模式使其代码逻辑小巧高效，因此在并发处理的能力上占据了绝对优势。

另外值得一提的是，Memcache 的内存 Hash 表所特有的 Expires 数据过期淘汰机制，正好和 Session 的过期机制不谋而合，这就降低了删除过期 Session 数据的代码复杂度。但对比“基于数据库的存储方案”，仅逻辑这块就给数据表带来了巨大的查询压力。

redis 作为 NoSQL 的后起之秀，经常被拿来与 Memcached 做对比。redis 作为一种缓存，或者干脆称之为 NoSQL 数据库，提供了丰富的数据类型（list、set 等），可以将大量数据的排序从单机内存解放到 redis 集群中进行处理，并且可以用于实现轻量级消息中间件。从性能上来说，redis 在对小于 100KB 的数据进行读写时，其速度优于 Memcached。在笔者的许多线上业务系统中，redis 已经取代 Memcached 来存放 Session 数据了。

7.2.3 什么是会话保持

会话保持并非 Session 共享。

在大多数的电子商务应用系统中，或者需要进行用户身份认证的在线系统中，一个客

户与服务器经常会经过好几次的交互过程才能完成一笔交易或一个请求。由于这几次交互过程是密切相关的，服务器在进行交互的过程中，要完成某一个交互步骤往往需要了解上一次交互的处理结果，或者上几步的交互结果，这就要求所有相关的交互过程都要由同一台服务器来完成，而不能被负载均衡器分散到不同的服务器上。

而这一系列相关的交互过程可能是由客户到服务器的一个连接的多次会话来完成的，也可能是在客户与服务器之间的多个不同连接里的多次会话来完成的。关于不同连接的多次会话，最典型的例子就是基于 HTTP 的访问，一个客户完成一笔交易可能需多次点击，而一个新的点击产生的请求，可能会重用上一次点击建立起来的连接，也可能是一个新建的连接。

会话保持就是指在负载均衡器上有这么一种机制，可以识别客户与服务器之间交互过程的关联性，在做负载均衡的同时，还能保证一系列相关连的访问请求被分配到同一台服务器上。

7.3 负载均衡器的会话保持机制

会话保持机制的目的是保证在一定时间内某一个用户与系统之间的会话只交给同一台服务器进行处理，这一点在满足网银、网购等应用场景的需求时格外重要。负载均衡器实现会话保持一般会有如下几种方案。

- ❑ 基于源 IP 地址的持续性保持：主要用于四层负载均衡，这种方案应该是大家最为熟悉的，LVS/HAProxy、Nginx 都有类似的处理机制，比如 Nginx 有 `ip_hash` 算法，HAProxy 有 `source` 算法。
- ❑ 基于 Cookie 数据的持续性保持：主要用于七层负载均衡，以便确保同一会话的报文能够被分配到同一台服务器中。其中，根据服务器的应答报文中是否携带含有服务器信息的 `Set-Cookie` 字段，又可以分为 Cookie 插入保持和 Cookie 截取保持。
- ❑ 基于 HTTP 报文头的持续性保持：主要用于七层负载均衡，当负载均衡器接收到某一个客户端的首次请求时，会根据 HTTP 报文头关键字建立持续性表项，记录下为该客户端分配的服务器情况，在会话表项的生存期内，后续具有相同 HTTP 报文头信息的连接都将发往该服务器进行处理。

7.3.1 LVS 的会话保持机制

LVS 是利用配置文件里的 `persistence`（单位为秒）设置来设定会话保持时间的，这个选项对于电子商务网站来说尤其有用：当用户从远程用账号登录网站时，有了这个会话保持功能，就能把用户的请求转发给同一个应用服务器了。在这里我们来做一个假设，假定现在有一个 LVS 环境，使用 VS/DR 转发模式，真实的 Web 服务器有 2 个，假设 LVS 负载均衡器不启用会话保持功能。当用户第一次访问的时候，他的访问请求被负载均衡器转给某个真实的服务器，这样他看到一个登录页面，第一次访问完毕；接着他在登录框里填写用户名和密码，然后提交；这时候，问题可能就会出现——登录不成功。因为没有会话保持，负载

均衡器可能会把第 2 次的请求转发到其他的服务器上，这样浏览器又会提醒客户需要再次输入用户名及密码。

在这里可以做一个简单的实验来验证一下，实验的 IP 分配如表 7-1 所示。

表 7-1 LVS 会话实验的服务器 IP 分配表

服务器名称	IP	用 途
LVS-Master	192.168.1.207	提供负载均衡
LVS-DR-VIP	192.168.1.210	集群 VIP 地址
Web1 服务器	192.168.1.205	提供 Web 服务
Web2 服务器	192.168.1.206	提供 Web 服务

这里的系统为 CentOS 6.4 x86_64，内核版本为 2.6.32-573.12.1.el6.x86_64。

由于笔者之前使用的是最小化安装，所以要先安装下编译工具等，另外为了不影响实验结果，建议关闭 iptables 防火墙和 SELinux，它们会直接影响实验结果；在后端的两台 Web 服务器上笔者直接安装了 httpd 服务，并分别设定了不同的首页地址，以示区分，安装基础的编译工具和 ipvsadm 软件需要的基础软件包的命令如下：

```
yum -y install gcc gcc-c++ kernel-devel libnl* libpopt* popt-static
```

其中，IPVS 是 LVS 的关键，因为 LVS 的 IP 负载均衡技术就是通过 IPVS 模块来实现的，IPVS 是 LVS 集群系统的核心软件，而 IPVS 具体是由 ipvsadm 来实现的。下面首先用命令查看下当前内核是否支持 IPVS，命令如下所示：

```
lsmod | grep ip_vs
```

结果发现是不支持的。

那么，需要在 LVS-MASTER 机器上安装 ipvsadm 软件，这里采用源码安装的方式，命令如下所示：

```
mkdir -p /usr/local/src
cd /usr/local/src
wget http://www.linuxvirtualserver.org/software/kernel-2.6/ipvsadm-1.26.tar.gz
tar xvf ipvsadm-1.26.tar.gz
cd ipvsadm-1.26
ln -s /usr/src/kernels/2.6.32-573.12.1.el6.x86_64/ /usr/src/linux
make
make install
```

安装成功以后输入 ipvsadm 命令验证，应该有如下显示：

```
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
-> RemoteAddress:Port Forward Weight ActiveConn InActConn
```

要确定是否有 ip_vs 模块，可输入如下命令验证：

```
lsmod | grep ip_vs
```

显示结果如下所示:

```
ip_vs          115643  0
libcrc32c      1246    1  ip_vs
ipv6           321422  16 ip_vs,ip6t_REJECT,nf_conntrack_ipv6,nf_defrag_ipv6
```

上面的结果表明 ip_vs 模块已成功加载, 然后编写并运行 lvs- initialization.sh 脚本, 其作用为绑定 VIP 地址到 LVS-MASTER 上, 并且设定 LVS 工作模式等, 脚本内容如下所示:

```
#!/bin/bash
VIP=192.168.1.210
RIP1=192.168.1.205
RIP2=192.168.1.206
. /etc/rc.d/init.d/functions

logger $0 called with $1
case "$1" in
    start)
        echo " Start LVS of DirectorServer"
        /sbin/ifconfig eth0:0 $VIP broadcast $VIP netmask 255.255.255.255 up
        /sbin/route add -host $VIP dev eth0:0
        echo "1" >/proc/sys/net/ipv4/ip_forward
        #Clear ipvsadm table
        /sbin/ipvsadm -C
        #Set LVS rules
        /sbin/ipvsadm -A -t $VIP:80 -s wrr -p 120
        #如果没有-p参数的话, 我们后续访问VIP地址时会发现, VIP地址会在后端的两台Web机器上轮询切换
        /sbin/ipvsadm -a -t $VIP:80 -r $RIP1:80 -g
        /sbin/ipvsadm -a -t $VIP:80 -r $RIP2:80 -g
        #Run LVS
        /sbin/ipvsadm
    ;;
    stop)
        echo "close LVS Directorserver"
        echo "0" >/proc/sys/net/ipv4/ip_forward
        /sbin/ipvsadm -C
        /sbin/ifconfig eth0:0 down
    ;;
    *)
        echo "Usage: $0 {start|stop}"
        exit 1
    esac
```

为脚本 lvs-initialization.sh 执行权限, 并执行它, 命令如下所示:

```
./lvs-initialization.sh start
```

脚本显示结果如下所示:

```
start LVS of DirectorServer
```

```

SIOCADDRT: File exists
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
-> RemoteAddress:Port      Forward Weight ActiveConn InActConn
TCP 192.168.1.210:http rr persistent 120
-> 192.168.1.205:http      Route 1 0 0
-> 192.168.1.206:http      Route 1 0 0

```

ActiveConn 表示活动连接数，即 TCP 连接状态的 ESTABLISHED；InActConn 表示其他非活动连接数，即所有的其他状态和 TCP 连接数。

最后，在后端的两台 Web 服务器上执行 realserver.sh 脚本，此脚本起的作用为：绑定 VIP 地址并设定 ARP 抑制。脚本 realserver.sh 的代码如下：

```

#!/bin/bash
SNS_VIP=192.168.1.210
. /etc/rc.d/init.d/functions

case "$1" in
start)
ifconfig lo:0 $SNS_VIP netmask 255.255.255.255 broadcast $SNS_VIP
/sbin/route add -host $SNS_VIP dev lo:0
echo "1" >/proc/sys/net/ipv4/conf/lo/arp_ignore
echo "2" >/proc/sys/net/ipv4/conf/lo/arp_announce
echo "1" >/proc/sys/net/ipv4/conf/all/arp_ignore
echo "2" >/proc/sys/net/ipv4/conf/all/arp_announce
sysctl -p >/dev/null 2>&1
echo "RealServer Start OK"
;;
stop)
ifconfig lo:0 down
route del $LVS_VIP >/dev/null 2>&1
echo "0" >/proc/sys/net/ipv4/conf/lo/arp_ignore
echo "0" >/proc/sys/net/ipv4/conf/lo/arp_announce
echo "0" >/proc/sys/net/ipv4/conf/all/arp_ignore
echo "0" >/proc/sys/net/ipv4/conf/all/arp_announce
echo "RealServer Stopped"
;;
*)
echo "Usage: $0 {start|stop}"
exit 1
esac
exit 0

```

分别在两台 Web 机器上执行脚本，命令如下所示：

```
./realserver.sh start
```

此脚本执行以后，就可以分别在两台 Web 机器上绑定 VIP 地址并设定 ARP 抑制了。

好了，现在来看看 LVS 持久连接技术，LVS 的持久性连接涉及两个方面（分别是保存时间和空闲超时时间）：

把同一个 Client 客户端的请求信息记录到 LVS 的哈希表里，保存时间使用 `persistence_timeout` (Keepalived 配置文件) 来控制，单位为秒。`persistence_granularity` 参数 (ipvsadm 里的 -M 参数) 是配合 `persistence_timeout` 的，在某些情况下特别有用，它的值是子网掩码，表示持久连接的粒度，默认是 255.255.255.255，也就是单独的客户端 IP，如果改成 255.255.255.0，则表示只要是一个网段的都会被分配到同一台后端 Web 机器上。

创建一个连接后的空闲超时时间分 3 种情况，分别如下：

- TCP 的空闲超时时间。
- LVS 收到客户端 `tcpfin` 的超时时间。
- UDP 的超时时间。

可以用如下命令来查看这些值：

```
ipvsadm -L --timeout
```

命令显示结果如下：

```
Timeout (tcp tcpfin udp): 900 120 300
```

显示结果分别对应了上面的 3 种情况。

我们用 `ipvsadm` 命令显示内核虚拟服务器表，命令如下所示：

```
ipvsadm -Lcn
```

显示结果如下所示：

```
IPVS connection entries
```

pro	expire	state	source	virtual	destination
TCP	01:30	FIN_WAIT	192.168.1.222:54446	192.168.1.210:80	192.168.1.206:80
TCP	01:39	NONE	192.168.1.222:0	192.168.1.210:80	192.168.1.206:80
TCP	01:39	FIN_WAIT	192.168.1.222:54500	192.168.1.210:80	192.168.1.206:80
TCP	01:32	FIN_WAIT	192.168.1.222:54460	192.168.1.210:80	192.168.1.206:80
TCP	01:39	FIN_WAIT	192.168.1.222:54488	192.168.1.210:80	192.168.1.206:80

当任意一个客户端访问 LVS 的 VIP 地址时，IPVS 都会记录一条状态为 `NONE` 的信息，`expire` 初始值是 `persistence_timeout` 的值，然后根据时钟主键变小，在以下记录存在期间，同一客户端 IP 连接上来后，都会被分配到同一个后端。

`FIN_WAIT` 的值就是 `tcpfin` 的超时时间，当 `NONE` 的值为 0 时，如果 `FIN_WAIT` 还存在，那么 `NONE` 的值会重新变成 60 秒，然后再减少，直到 `FIN_WAIT` 消失，`NONE` 才会消失，只要 `NONE` 存在，同一客户端的访问都会分配到统一的 Real Server 上面。这个说法很好验证，大家只要不停地执行 `ipvsadm -Lcn` 的命令，观察其时间变化即可。

参考文档和资料：

<http://www.linuxvirtualserver.org/docs/persistence.html>。

http://xstarcd.github.io/wiki/sysadmin/lvs_persistence.html。

7.3.2 Nginx 负载均衡器中的 ip_hash 算法

Nginx 作为负载均衡器时，其提供 upstream 模块的 ip_hash 机制，能够将某个 IP 的请求定向到同一台后端服务器上，这样一来这个 IP 下的某个客户端和某个后端服务器就能建立起稳固的连接了。ip_hash 算法可以看成是 roundrobin 算法的升级版，如果后端有某台 Web 机器出现故障，则 ip_hash 算法会自动降成 roundrobin，有兴趣的朋友可以自行测试。

ip_hash 是在 upstream 配置中定义的，脚本如下：

```
upstream backend {  
    ip_hash;  
    server 192.168.1.106:80;  
    server 192.168.1.107:80;  
}
```

笔者已在线上采用过 Nginx 的这种 ip_hash 算法机制，而且采用这种机制的网站一直运行稳定，即使是在并发量大的情况下也没有发生过 Session 丢失的现象，这就证明了这种技术的可靠性，特推荐给大家。

在没有专门用来存放 Session 的 Memcached 或 redis 机器的场景里，可以采用此 ip_hash 算法让客户始终只访问固定的后端 Web 机器，以解决 Session 共享的问题。

7.3.3 HAProxy 负载均衡器的 source 算法

HAProxy 也有和 Nginx 的 ip_hash 算法类似的机制，即 source 算法，它也可以实现会话保持功能；我们可以通过配置一个简单的 1+2 Web 架构来验证一下，此处的 IP 跟前面 LVS 中的一样，只不过这里没有了 VIP 的概念，详细过程如下。

(1) 安装 HAProxy

在此之前，要提前配置好 epel 外部 yum 源，步骤略过。

下面查看当前 yum 源是否提供了 HAProxy 的 rpm 包，命令如下：

```
yum list | grep haproxy
```

结果如下：

```
haproxy.x86_64                               1.5.4-2.el6_7.1                updates
```

现在，通过 yum 命令安装 HAProxy，命令如下：

```
yum install haproxy -y
```

(2) 修改 HAProxy 默认配置文件

记得不要采用它默认的轮询方式 (roundrobin)，而是要采用 source。配置文件 /etc/haproxy/haproxy.cfg 的内容如下：

```
global  
    log                          127.0.0.1 local3  
    chroot                      /var/lib/haproxy
```



```

pidfile /var/run/haproxy.pid
maxconn 4000
user haproxy
group haproxy
daemon
stats socket /var/lib/haproxy/stats

```

```

defaults
mode http
log global
option httplog
option dontlognull
option http-server-close
option forwardfor except 127.0.0.0/8
option redispatch
retries 3
timeout http-request 10s
timeout queue 1m
timeout connect 10s
timeout client 1m
timeout server 1m
timeout http-keep-alive 10s
timeout check 10s
maxconn 3000

```

```

listen stats #这里定义的是HAProxy监控
mode http #模式HTTP
bind 0.0.0.0:1080 #绑定的监控IP与端口
stats enable #启用监控
stats hide-version #隐藏HAProxy版本
stats uri /web_status #定义的URI
stats realm Haproxy\ Statistics #定义显示文字
stats auth admin:admin #认证

```

```

frontend http
bind *:80
mode http
log global
option logasap
option dontlognull
capture request header Host len 20
capture request header Referer len 20
default_backend web

```

```

backend web
balance source
server web1 192.168.1.205:80 check maxconn 2000
server web2 192.168.1.206:80 check maxconn 2000

```

新版的 HAProxy 支持以服务 reload 的模式启动，这样更改配置文件以后，就可以用如

下命令来重载 HAProxy 服务了：

```
service haproxy reload
```

HAProxy 的配置中包含如下 5 个组件，当然这些组件不是必选的，可以根据需要选择配置。

- ❑ Global：参数是进程级的，通常和操作系统相关。这些参数一般只设置一次，如果配置无误，则不需要再次配置了。
- ❑ Defaults：配置默认参数，这些参数可以配置到 Frontend、Backend、Listen 组件中。
- ❑ Frontend：接收请求的前端虚拟节点，Frontend 可以根据规则直接指定具体使用的后端 Backend（可动态选择）。
- ❑ Backend：后端服务集群的配置，是真实的服务器，一个 Backend 对应一个或多个实体服务器。
- ❑ Listen：Frontend 和 Backend 的组合体。

关于 HAProxy 的详细配置文件说明参数请大家参考附录 A，请注意版本之间的差异性变化。

（3）启动 HAProxy

新版 HAProxy 支持 reload 命令，启动之前先检查下配置文件有无语法方面的问题，命令如下：

```
haproxy -f /etc/haproxy/haproxy.conf -c
```

结果显示如下：

```
Configuration file is valid
```

然后启动 HAProxy，命令如下：

```
service haproxy start
```

HAProxy 自带了强大的监控功能，在网址 http://192.168.1.207:1080/web_status/ 中输入相应的账号和密码就可以看到监控页面，监控页面如图 7-5 所示。

HAProxy

Statistics Report for pid 13510

> General process information

pid = 13510 (process #1, nproc = 1)
uptime = 0d 0h00m51s
system limits: memmax = unlimited, ulimit-n = 8034
maxsock = 8034, maxconn = 4000, maxpipes = 0
current conns = 3, current pipes = 0/0, conn rate = 3/sec
Running tasks: 1/10, idle = 100 %

active UP
active UP, going down
active DOWN, going up
active or backup DOWN
active or backup DOWN for maintenance (MAINT)
active or backup SOFT STOPPED for maintenance
Note: "NOLEBDRAIN" = UP with load-balancing disabled.

backup UP
backup UP, going down
backup DOWN, going up
not checked

Display option:
• Scope:
• Hide DOWN servers
• Refresh now
• CSV export

External resources:
• Primary site
• Updates (v1.5)
• Online manual

stats

	Queue			Session rate			Sessions				Bytes				Denied			Errors			Warnings			Status	Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	LastChk	Wght		Act	Bck	Chk	Dwn	Downtime	Thrtle		
Frontend	0	0	0	0	0	0	3	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Backend	0	0	0	0	0	0	0	0	0	300	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

http

	Queue			Session rate			Sessions				Bytes				Denied			Errors			Warnings			Status	Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	LastChk	Wght		Act	Bck	Chk	Dwn	Downtime	Thrtle		
Frontend	0	0	0	0	0	0	3	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

web

	Queue			Session rate			Sessions				Bytes				Denied			Errors			Warnings			Status	Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	LastChk	Wght		Act	Bck	Chk	Dwn	Downtime	Thrtle		
web1	0	0	0	0	0	0	0	3	0	1	2000	4	4	115	1990	5929	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
web2	0	0	0	0	0	0	0	0	0	0	2000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Backend	0	0	0	0	0	0	0	3	0	1	3000	4	4	115	1990	5929	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

图 7-5 HAProxy 的监控页面

(4) 设置 HAProxy 的日志配置策略

在默认情况下,为了节省读写 I/O 所消耗的性能,HAProxy 没有自动配置日志输出功能,但线上的生产环境有时为了维护和调试方便,是需要有日志输出的,所以我们可以根据需求来配置 HAProxy 的日志配置策略。

首先,设置 HAProxy 的默认配置文件跟日志相关的选项,命令如下所示:

```
global
    log                127.0.0.1 local3 #local3相当于info级别的日志
    chroot             /var/lib/haproxy
    pidfile            /var/run/haproxy.pid
    maxconn            4000
    user               haproxy
    group              haproxy
    daemon
    stats socket /var/lib/haproxy/stats
```

然后,编辑系统日志配置 /etc/rsyslog.conf,此文件默认会读取 /etc/rsyslog.d/*.conf 目录下的配置文件,所以我们可以将 HAProxy 的相关配置放在其下,这里取名为 haproxy.conf,文件内容如下:

```
$ModLoad imudp
$UDPServerRun 514
local3.* /var/log/haproxy.log
```

其中,imudp 是模块名,支持 UDP 协议。\$UDPServerRun 514 表示允许 514 端口接收使用 UDP 和 TCP 协议转发过来的日志,而 rsyslog 在默认情况下,正是在 514 端口监听 UDP 的。local3 相当于 info 级别的日志,/var/log/haproxy.log 后面跟的是详细路径名。

最后,修改 /etc/sysconfig/rsyslog 文件,修改内容如下:

```
# Options for rsyslogd
# Syslogd options are deprecated since rsyslog v3.
# If you want to use them, switch to compatibility mode 2 by "-c 2"
# See rsyslogd(8) for more details
SYSLOGD_OPTIONS="-c 2 -r -m 0"
```

其中,各参数的作用分别如下:

- -c 指定运行兼容模式。
- -r 接收远程日志。
- -x 在接收客户端消息时,禁用 DNS 查找。需和 -r 参数配合使用。
- -m 标记时间戳。单位是分钟,为 0 时,表示禁用该功能。

现在,来看看 HAProxy 的日志内容:

```
Jan 11 06:50:00 localhost haproxy[13637]: 192.168.1.222:49629 [11/
Jan/2016:06:49:58.136] http web/web1 1922/0/0/0/+1922 403 +177 - - ----
3/3/1/1/0 0/0 {192.168.1.207|} "GET / HTTP/1.1"
Jan 11 06:50:00 localhost haproxy[13637]: 192.168.1.222:49629 [11/
```

```

Jan/2016:06:49:58.136] http web/web1 1922/0/0/0/+1922 403 +177 - - ----
3/3/1/1/0 0/0 {192.168.1.207|} "GET / HTTP/1.1"
Jan 11 06:50:00 localhost haproxy[13637]: 192.168.1.222:49629 [11/
Jan/2016:06:50:00.057] http web/web1 31/0/0/0/+31 304 +130 - - ---- 3/3/1/1/0
0/0 {192.168.1.207|http://192.168.1.207} "GET /icons/apache_pb.gif HTTP/1.1"
Jan 11 06:50:00 localhost haproxy[13637]: 192.168.1.222:49629 [11/
Jan/2016:06:50:00.057] http web/web1 31/0/0/0/+31 304 +130 - - ---- 3/3/1/1/0
0/0 {192.168.1.207|http://192.168.1.207} "GET /icons/apache_pb.gif HTTP/1.1"
Jan 11 06:50:00 localhost haproxy[13637]: 192.168.1.222:49629 [11/
Jan/2016:06:50:00.057] http web/web1 31/0/0/0/+31 304 +130 - - ---- 3/3/1/1/0
0/0 {192.168.1.207|http://192.168.1.207} "GET /icons/apache_pb.gif HTTP/1.1"
Jan 11 06:50:00 localhost haproxy[13637]: 192.168.1.222:49640 [11/
Jan/2016:06:49:57.738] http web/web1 2355/0/0/0/+2355 304 +130 - - ----
3/3/1/1/0 0/0 {192.168.1.207|http://192.168.1.207} "GET /icons/poweredby.png
HTTP/1.1"
Jan 11 06:50:00 localhost haproxy[13637]: 192.168.1.222:49640 [11/
Jan/2016:06:49:57.738] http web/web1 2355/0/0/0/+2355 304 +130 - - ----
3/3/1/1/0 0/0 {192.168.1.207|http://192.168.1.207} "GET /icons/poweredby.png
HTTP/1.1"
Jan 11 06:50:00 localhost haproxy[13637]: 192.168.1.222:49640 [11/
Jan/2016:06:49:57.738] http web/web1 2355/0/0/0/+2355 304 +130 - - ----
3/3/1/1/0 0/0 {192.168.1.207|http://192.168.1.207} "GET /icons/poweredby.png
HTTP/1.1"
Jan 11 06:50:10 localhost haproxy[13637]: 192.168.1.222:49640 [11/
Jan/2016:06:50:00.092] http web/web1 9959/0/0/0/+9959 403 +177 - - ----
2/2/1/1/0 0/0 {192.168.1.207|} "GET / HTTP/1.1"
Jan 11 06:50:10 localhost haproxy[13637]: 192.168.1.222:49640 [11/
Jan/2016:06:50:00.092] http web/web1 9959/0/0/0/+9959 403 +177 - - ----
2/2/1/1/0 0/0 {192.168.1.207|} "GET / HTTP/1.1"
Jan 11 06:50:10 localhost haproxy[13637]: 192.168.1.222:49640 [11/
Jan/2016:06:50:00.092] http web/web1 9959/0/0/0/+9959 403 +177 - - ----
2/2/1/1/0 0/0 {192.168.1.207|} "GET / HTTP/1.1"

```

从日志内容可发现，HAProxy 采用了 source 算法以后，无论怎么刷新，通过前面的 HAProxy LB 机器始终只能访问到后端的 Web1 机器上面。在没有 Session 的 Memcached 或 redis 机器的场景里，可以通过采用此 source 算法，让客户始终只访问固定的后端 Web 机器，以此来解决 Session 共享的问题。

在项目实施中，我会根据客户的需求，将 HAProxy 用于一些时效性强的中小型网站上（比如金融证券类的新闻资讯网站），做成基于单机 HAProxy（后面接两台 Web 机器）的网

站，因为这些网站只是在早上 9:00 到下午 5:00 之间会有用户访问，鉴于 HAProxy 的稳定性、接近硬件设备的网络吞吐量，以及其所拥有的强大监控功能，其完全可以胜任这项工作。如果大家也有这种需求，不妨考虑一下这种 1+2 型的做法。

7.3.4 服务器健康检测技术

负载均衡器现在都使用了非常多的服务器健康检测技术，主要方法是通过发送不同类型的协议包然后检查能否接收到正确的应答来判断后端的服务器是否存活，如果后端的服务器出现故障就会自动剔除。其中所包括的三种主要技术如下。

- ❑ ICMP：负载均衡器向后端的服务器发送 ICMP ECHO 包（就是我们俗称的“ping”），如果能正确收到 ICMP REPLY，则证明服务器 ICMP 协议处理正常，即服务器是活着的。
- ❑ TCP：负载均衡器向后端的某个端口发起 TCP 连接请求，如果成功完成三次握手，则证明服务器 TCP 协议处理正常。
- ❑ HTTP：负载均衡器向后端的服务器发送 HTTP 请求，如果收到的 HTTP 应答内容是正确的，则证明服务器 HTTP 协议处理正常。

这里以 Nginx 负载均衡器来举例说明下。

upstream 模块是 Nginx 负载均衡器的主要模块，它提供了简单的办法来实现轮询和客户端 IP 之间的后端服务器负载均衡，并且可以对服务器进行健康检查。upstream 并不处理请求，而是通过请求后端服务器得到用户的请求内容。在转发给后端时，默认是轮询。下面为一组服务器负载均衡的集合：

```
upstream php_pool {
    server 192.168.1.7:80 max_fails=2 fail_timeout=5s;
    server 192.168.1.8:80 max_fails=2 fail_timeout=5s;
    server 192.168.1.9:80 max_fails=2 fail_timeout=5s;
}
```

这里针对 upstream 模块的相关指令进行一下说明。

- ❑ max_fails：定义可以发生错误的最大次数。
- ❑ fail_timeout：若 Nginx 在 fail_timeout 设定的时间内与后端服务器通信失败的次数超过 max_fails 设定的次数，则认为这个服务器不再起作用；在接下来的 fail_timeout 时间内，Nginx 不再将请求分发给该失效的机器。
- ❑ down：把后端标记为离线，仅限于 ip_hash。
- ❑ backup：把后端标记为备份服务器，在后端服务器全部无效时才启用。

Nginx 的健康检查主要是针对后端服务所提供的，且功能被集成在 upstream 模块中，共有如下两个指令：max_fails 和 fail_timeout，指令说明见上文。

Nginx 健康检查机制为：在检测到后端服务器故障后，Nginx 依然会把请求转向该服务器，不过，当它发现 timeout 或 refused 时，则会把请求改发到 upstream 的其他节点，直到

获得正常数据后, Nginx 才会把数据返回给用户, 这也体现了 Nginx 的异步传输。这一点跟 LVS/HAProxy 的区别很大, 在 LVS/HAProxy 里, 每个请求都只有一次机会, 假如用户发起一个请求, 结果该请求分到的后端服务器刚好挂掉了, 那么这个请求就失败了。

7.4 Linux 集群的项目案例分享

7.4.1 案例分享一：用 Nginx+Keepalived 实现在线票务系统

这是一套以前实施过的在线订票系统, 防火墙、交换机、服务器均置于电信机房的同一机柜中, 服务器的带宽为 40MB, 项目拓扑图如图 7-6 所示。

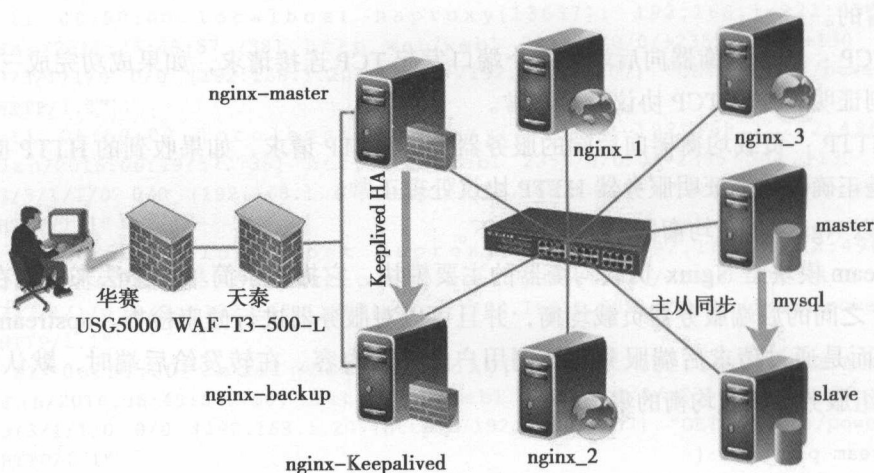


图 7-6 在线订票系统网络拓扑图

1. 整套系统的安全考虑

因为牵涉到信用卡和银行卡在线支付的问题, 所以系统的安全性就需要尤为重视了。在项目实施的过程中, 在安全防护方面, 笔者公司是采用硬件防火墙加上应用层防火墙双层防护来实现的, 系统均安装了 CentOS 5.4 x86_64; 软件层分为负载均衡层、Web 层、数据库层, 整套系统均关闭了 iptables 防火墙, 只映射 Keepalived 虚拟的 VIP 在最前端的华赛 USG5000 的外网 80 端口上, 先将整套系统的安全级别提高到金融安全级别, 再考虑负载均衡及其他事宜。另外, 网络工程师都应该清楚, 防火墙有三种工作模式——路由、透明和混合模式, 在这里, 华赛 USG5000 防火墙用的是路由模式, 而天泰防火墙用的则是透明模式。

下面简单介绍下这两种防火墙。

华赛 USG5000 可以有效抵御高强度的网络攻击, 同时还可以保证正常的网络应用。其基于多核处理器的硬件构架, 依靠多线程处理设计提供了十分优异的数据处理能力, 完全可以为 Internet 服务提供商、大型企业、园区网、数据中心等具备大流量网络带宽的用户提

供高性能的安全防御手段。尤其是 USG5000 所具备的超高的“每秒新建连接数”，不仅可以对多种并行的网络应用实现快速响应，而且在大流量网络攻击的情况下，仍然可以防止网络业务中断，避免给用户带来损失。

USG5000 能有效地保障网络的正常运行，并且其独特的 GTP 安全防护功能可以为 GPRS 网络提供有效的安全防护。USG5000 安全网关可以抵御大流量的 DDoS 攻击，为用户的业务系统提供 DDoS 攻击防护，依靠其优越的产品性能，能防范每秒数百万包以上的 DDoS 攻击，可准确识别并控制 SYN FLOOD、UDP FLOOD、ICMP FLOOD、DNS FLOOD、CC 等多种 DDoS 攻击，同时还能提供蠕虫病毒流量的识别和防范功能，结合华为赛门铁克专有的 ICA 智能连接算法，保证在准确识别 DDoS 攻击流量的同时，不影响用户的正常访问，在复杂的网络情况下实现真正的安全防护，是业界领先的 DDoS 防护设备，上述功能也是我们关注的重点。

而天泰 WAF-T3-500-L 安全网关防火墙则具备全面的攻击防御系统，可保证系统不受网络蠕虫/病毒和应用专用漏洞的攻击，并且大大缓解了来自网络层和应用层 DoS/DDoS 攻击的影响。网关的 NetShield 引擎在网络层会对数据进行细致检查，彻底阻断来自网络层的潜在攻击，并在应用层会对 Web 请求进行检查，辨别恶意内容并阻止其进入应用服务器。

WAF-T3-500-L 安全性能如下。

- 常见网络攻击防护：保护网络基础设施不受常见的、来自网络层的恶意攻击的影响。
- DoS/DDoS 保护：识别网络层和应用层的 DoS/DDoS 攻击，缓解攻击对应用基础设施的影响（这也是我们关注的重点）。
- 入侵过滤：通过在恶意蠕虫和病毒进入应用服务器之前进行识别并拒绝其进入，保护应用服务器不受侵袭。
- SSL 加密：应用内容在传输过程中都会受到加密保护，通过转移服务器复杂的加/解密任务将应用处理能力发挥到极致。该功能可保护敏感应用内容的安全，使其摆脱被窃取及被滥用的潜在威胁。

此外，能实现 SQL 注入攻击、钓鱼攻击、跨站脚本攻击的防护，以及常见的系统溢出的防护等，这也是我们比较关注的内容。

关于安全证书的配置，笔者公司购买了 Geo Trust 的商业证书，它是支持多域名的 HTTPS 的，防止以后域名有 rewrite 跳转的需求，价格自然也不菲了。另外，我们还购买了 McAfee 的网站扫描服务，这一服务是针对代码层面安全的。

2. 硬件方面的投入

我们一般采用的服务器是 HP DL380G6（用于后面的 Web 集群）和 HP DL580G5（用于 MySQL 数据库）。在项目实施过程中我们发现 HP DL580G5 的性能确实优越，如果成本充足，可考虑采用此服务器作为应用服务器。它跟以往的老型号不同，用的是双四核至强 E7440 3.2G 的 CPU，内存一般是 64GB 或 146GB，这可以根据项目成本来权衡。在租用机

房时，我们一般选择的是电信机房，也可以考虑北京双线通机房；出口带宽建议为 40MB。

3. 负载均衡层

在负载均衡层，我们采用的软件是 Nginx 0.8.15 源码（当时最新最稳定的版本），两台 Nginx 负载均衡用 Keepalived 作高 HA。其实也可以用 LVS/Keepalived 来实现，但我们在项目实施过程中发现，Nginx 在正则处理及分发上效果比 LVS 更好（有些功能 LVS 实现不了），而且稳定性也不错。在已上线的金融资讯类网站的项目里，笔者做的是 1+2 的架构（按客户的要求），已经稳定运行好几年了，当然也要配合 Cacti+Nagios 进行实时监控。

那么如何处理 Session 的问题呢？有如下两种方案。

- ❑ Nginx 负载均衡器采用 ip_hash 算法机制，让访问的客户端始终与后端的某台 Web 服务器建立固定的连接关系。
- ❑ 采用与 PHPCMS 类似的方法，将 Session 写进后端的统一数据库里，例如 MySQL 数据库。

前期我们采用的是第二种方案，后来发现数据库的压力会因此而增大，所以采取了前一种会话保持的方法。

4. Web 层

页面同步的办法如下所示：

- ❑ 不同的 Web 服务器之间，代码的同步可以采用 rsync+inotify 的办法，图片建议用独立的存储。
- ❑ 后端采用共用存储，读数据采用同一个存储设备。这里说一下要用到的存储设备，我们用得比较多的是 EMC CLARiiON CX4 的 FC 磁盘阵列，它很稳定，没发生过丢失数据库的问题；缺点是比较贵，会增加整个系统的实施成本。
- ❑ 在 PHP 程序上实现动态地中调取数据，不在 Web 服务器中调取而直接采用后端的文件服务器或存储。

前期笔者公司用的是单 NFS 方案，后期采用的是 DRBD+Heartbeat+NFS 方案（客户最后要求要上存储，我们就上了 EMC CLARiiON CX4），其稳定性还是很不错的。

个人觉得这几种方案里性价比最高的当属 rsync+inotify 了。

此外，Web 集群方面用的是 Nginx+PHP5 (FastCGI)，这里说一下并发的問題。在设计项目方案时，我们考虑单台 Web 服务器上的并发值为 3000，在局域网环境中（要考虑网络环境的影响）通过 LoadRunner 反复测试，单台 Nginx 的 Web 服务器通过 3000 的并发是没有问题的，3 台 Web 机器就是 3000×3 并发。但系统正式上线时会发现，在非游戏类的网站上根本达不到 9000 并发，这只是一个理论值。本着高扩展性的原则，我们还是尽量在硬件和性能上对单台 Web 服务器进行了调优。我们要设计的这个票务系统是 9000 万张票，预计并发在 2000 左右，此系统架构完全能胜任此并发情况。另外，Nginx 作为负载均衡器 / 代理服务器在高并发下的稳定性是毋庸置疑的，有相关项目经验的人都应该清楚这点。

5. 数据库层

考虑到数据库层的压力情况，这里提出四种设计方案：

- ❑ 采用最常用的 MySQL 一主一从方案，在主 MySQL 数据库上做好单机数据库的优化。
- ❑ 采用 MySQL 的一主多从、读写分离方案，另还可以考虑自己开发中间件技术，让真正实现写功能的 MySQL 压力降低，从而达到数据库架构级调优的功能。
- ❑ 可以做 MySQL 数据库的垂直切分，将压力过大的 MySQL 数据库根据业务分成几个小数据库，以减轻压力。
- ❑ 如果读写压力还是过大，考虑采用 Oracle 数据库的 RAC 方案，我们曾用此方案成功解决了某企业 100 万用户的 OA 在线系统数据库压力大的问题，当然预算成本也大大增加了。

项目实施时，我们用的是 MySQL 一主一从方案，项目上线后发现事实上数据库的压力并没有想象中的那么大。

目前整套系统已在线上稳定运行，并且在高并发时间段也没有发生任何问题。笔者通过设计这套网站架构（包括防火墙的型号）整理出了一个框架，并已形成工作文档，可用于公司其他项目的实施方案中。

7.4.2 案例分享二：企业级 Web 负载均衡高可用之 Nginx+Keepalived

推荐掌握企业级的成熟的 Nginx+Keepalived 负载均衡高可用方案，其拓扑图如图 7-7 所示。

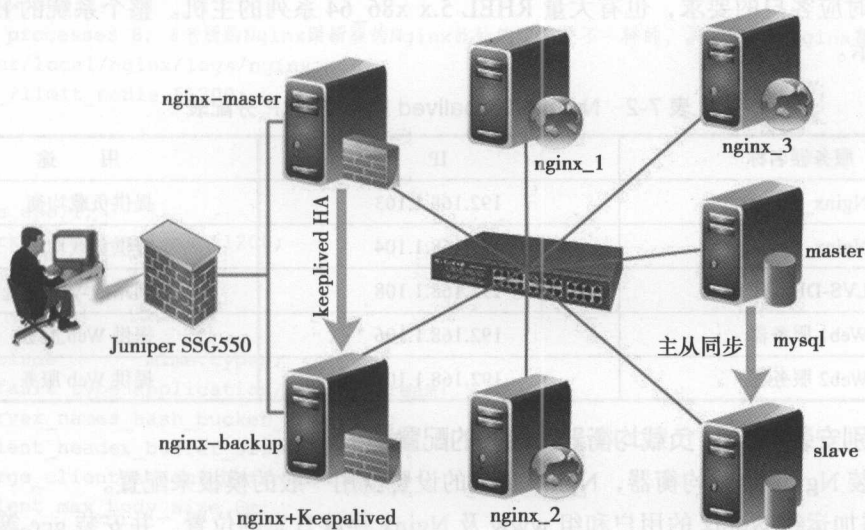


图 7-7 Nginx+Keepalived 负载均衡高可用网络拓扑图

一般为了维护方便，企业网站的服务器都在自己的内部机房里，只开放了 Keepalived

的 VIP 地址的两个端口 80、443，通过 Juniper SSG550 防火墙映射出去，外网 DNS 对应映射后的公网 IP。此架构的防火墙及网络安全说明如下。

此系统架构仅映射内网 VIP 的 80 及 443 端口于外网的 Juniper SSG550 防火墙下，其他端口均关闭，内网的所有机器均关闭 iptables 及 ipfw 防火墙；外网 DNS 指向通过 Juniper 或华赛 USG5000 映射出来的外网地址。

本节内容出自笔者的项目方案，这种负载均衡的方式同时也应用于笔者公司的电子商务网站中，目前已经稳定上线一年多了。通过下面的内容，大家可以迅速地架构起一个企业级的负载均衡高可用的 Web 环境。在负载均衡高可用技术上，笔者一直推崇以 Nginx+Keepalived 做 Web 的负载均衡高可用架构，并积极将其应用于真实项目中，此架构极适合灵活稳定的环境。Nginx 负载均衡作服务器遇到的故障一般有：

- ❑ 服务器网线松动等网络故障。
- ❑ 由于服务器硬件故障而导致宕机。
- ❑ Nginx 服务死掉。

遇到前两种情况时，Keepalived 是能起到 HA 的作用的；然而遇到第三种情况时就没有办法了，但可以通过 Shell 脚本监控解决这问题，从而实现真正意义上的负载均衡高可用。笔者在电子商务网站上就采用了这种方法，下面将其安装步骤详细说明一下。

1. Nginx+Keepalived 的说明及环境说明

关于服务器系统，从早期的 CentOS 5.1 x86_64 到现在的 CentOS 5.5 x86_64 我们均有涉及，有时应客户的要求，也有大量 RHEL 5.x x86_64 系列的主机。整个系统的 IP 情况如表 7-2 所示。

表 7-2 Nginx+Keepalived 服务器的 IP 分配表

服务器名称	IP	用 途
Nginx_Master	192.168.1.103	提供负载均衡
Nginx_Backup	192.168.1.104	提供负载均衡
LVS-DR-VIP	192.168.1.108	网站的 VIP 地址
Web1 服务器	192.168.1.106	提供 Web 服务
Web2 服务器	192.168.1.107	提供 Web 服务

2. 分别安装 Nginx 负载均衡器及相关的配置脚本

先安装 Nginx 负载均衡器，Nginx 负载的设置就用一般的模板来配置。

1) 添加运行 Nginx 的用户和组 www 及 Nginx 存放日志的位置，并安装 gcc 等基础库，以免发生 libtool 报错现象，命令如下：

```
yum -y install gcc gcc+ gcc-c++ openssl openssl-devel
groupadd www
useradd -g www www
```



```
mkdir -p /data/logs/
chown -R www:www /data/logs/
```

2) 下载并安装 Nginx 0.8.15 (当时最新最稳定的版本), 另外建议在工作中养成良好的习惯, 下载的软件包均放到 /usr/local/src 下, 命令如下:

```
cd /usr/local/src
wget http://blog.s135.com/soft/linux/nginx_php/pcpre/pcpre-7.9.tar.gz
tar zxvf pcpre-7.9.tar.gz
cd pcpre-7.9/
./configure
make && make install
cd ../
wget http://sysoev.ru/nginx/nginx-0.8.15.tar.gz
tar zxvf nginx-0.8.15.tar.gz
cd nginx-0.8.15/
./configure --user=www --group=www --prefix=/usr/local/nginx --with-http_stub_
status_module --with-http_ssl_module
make && make install
cd ../
```

配置 Nginx 负载均衡器的配置文件是 vim /usr/local/nginx/conf/nginx.conf, 下面的配置文件仅仅是笔者某项目的配置文档, 纯 HTTP 转发; 如果要添加 SSL 支持也很简单, 后面会有相关的说明, 记得将购买的 GeoTrust 证书文件放到 Nginx 负载均衡器上 (两台 Nginx 机器均要放), 而非置于后面的 Web 机器上, 配置文件内容如下:

```
user www www;
worker_processes 8; #老版的Nginx跟新版的Nginx此处的配置是不一样的, 具体请参考Nginx官方文档
pid /usr/local/nginx/logs/nginx.pid;
worker_rlimit_nofile 51200;

events
{
    use epoll;
    worker_connections 51200;
}

http{
    include mime.types;
    default_type application/octet-stream;
    server_names_hash_bucket_size 128;
    client_header_buffer_size 32k;
    large_client_header_buffers 4 32k;
    client_max_body_size 8m;
    sendfile on;
    tcp_nopush on;
    keepalive_timeout 60;
    tcp_nodelay on;
    fastcgi_connect_timeout 300;
    fastcgi_send_timeout 300;
```

```

fastcgi_read_timeout 300;
fastcgi_buffer_size 64k;
fastcgi_buffers 4 64k;
fastcgi_busy_buffers_size 128k;
fastcgi_temp_file_write_size 128k;
gzip on;
gzip_min_length 1k;
gzip_buffers 4 16k;
gzip_http_version 1.0;
gzip_comp_level 2;
gzip_types text/plain application/x-javascript text/css application/xml;
gzip_vary on;

upstream backend {
    ip_hash;
    server 192.168.1.106:80;
    server 192.168.1.107:80;
}

server {
    listen 80;
    server_name www.lpaituan.com;
    location / {
        root /var/www/html ;
        index index.php index.htm index.html;
        proxy_redirect off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass http://backend;
    }

    location /nginx {
        access_log off;
        auth_basic "NginxStatus";
        #auth_basic_user_file /usr/local/nginx/htpasswd;
    }

    log_format access '$remote_addr - $remote_user [$time_local] "$request" '
        '$status $body_bytes_sent "$http_referer" '
        '"$http_user_agent" $http_x_forwarded_for';
    access_log /data/logs/access.log access;
}
}

```

分别在两台 Nginx 负载均衡器上执行 `/usr/local/nginx/sbin/nginx` 命令，启动 Nginx 进程，然后用如下命令来检查：

```
lsuf -i:80
```

此命令显示结果如下：

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE	NODE	NAME
---------	-----	------	----	------	--------	------	------	------

nginx	13875	root	6u	IPv4	25918	TCP	*:http	(LISTEN)
nginx	13876	www	6u	IPv4	25918	TCP	*:http	(LISTEN)
nginx	13877	www	6u	IPv4	25918	TCP	*:http	(LISTEN)
nginx	13878	www	6u	IPv4	25918	TCP	*:http	(LISTEN)
nginx	13879	www	6u	IPv4	25918	TCP	*:http	(LISTEN)
nginx	13880	www	6u	IPv4	25918	TCP	*:http	(LISTEN)
nginx	13881	www	6u	IPv4	25918	TCP	*:http	(LISTEN)
nginx	13882	www	6u	IPv4	25918	TCP	*:http	(LISTEN)
nginx	13883	www	6u	IPv4	25918	TCP	*:http	(LISTEN)

Nginx 程序正常启动后，两台 Nginx 负载均衡器就算安装成功了，现在我们要安装 Keepalived 来实现这两台 Nginx 负载均衡器的高可用。

3. 安装 Keepalived，让其分别做 Web 及 Nginx 的 HA

1) 安装 Keepalived，并将其做成服务模式，以方便后续调试，Keepalived 的安装方法如下：

```
wget http://www.keepalived.org/software/keepalived-1.1.15.tar.gz
tar zxvf keepalived-1.1.15.tar.gz
cd keepalived-1.1.15
./configure --prefix=/usr/local/keepalived
make
make install
```

2) 安装成功后做成服务模式，方便启动和关闭，方法如下：

```
cp /usr/local/keepalived/sbin/keepalived /usr/sbin/
cp /usr/local/keepalived/etc/sysconfig/keepalived /etc/sysconfig/
cp /usr/local/keepalived/etc/rc.d/init.d/keepalived /etc/init.d/
```

3) 接下来就是分别设置主和备 Nginx 上的 Keepalived 配置文件了。先配置主 Nginx 上的 keepalived.conf 文件，如下所示：

```
mkdir /etc/keepalived
cd /etc/keepalived/
```

可以用 vim 编辑 /etc/keepalived.conf，内容如下：

```
! Configuration File for keepalived
global_defs {
    notification_email {
        yuhongchun027@163.com
    }
    notification_email_from keepalived@chtopnet.com
    smtp_server 127.0.0.1
    smtp_connect_timeout 30
    router_id LVS_DEVEL
}

vrrp_instance VI_1 {
    state MASTER
    interface eth0
```

```

virtual_router_id 51
mcast_src_ip 192.168.1.103 #mcast_src_ip此处是发送多播包的地址，如果不设置，则默认使
    用绑定的网卡
priority 100 #此处的priority是100，注意跟backup机器的区分
advert_int 1
authentication {
    auth_type PASS
    auth_pass chtopnet
}
virtual_ipaddress {
    192.168.1.108
}
}

```

下面设置备用 Nginx 上的 keepalived.conf 的配置文件，注意将其与主 Nginx 上的 keepalived.conf 区分开，代码如下：

```

! Configuration File for keepalived
global_defs {
    notification_email {
        yuhongchun027@163.com
    }
    notification_email_from keepalived@chtopnet.com
    smtp_server 127.0.0.1
    smtp_connect_timeout 30
    router_id LVS_DEVEL
}

vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    mcast_src_ip 192.168.1.104
    priority 99
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass chtopnet
    }
    virtual_ipaddress {
        192.168.1.108
    }
}

```

在两台负载均衡器上分别启动 Keepalived 程序，命令如下：

```
service keepalived start
```

下面来看一下主 Nginx 机器上与 Keepalived 相关的日志，可用如下命令查看：

```
tail /var/log/messages
```

此命令显示结果如下：

```
May 6 05:10:42 localhost Keepalived_vrrp: Configuration is using : 62610 Bytes
May 6 05:10:42 localhost Keepalived_vrrp: VRRP sockpool: [ifindex(2), proto(112), fd(8,9)]
May 6 05:10:43 localhost Keepalived_vrrp: VRRP_Instance(VI_1) Transition to
MASTER STATE
May 6 05:10:44 localhost Keepalived_vrrp: VRRP_Instance(VI_1) Entering MASTER
STATE
May 6 05:10:44 localhost Keepalived_vrrp: VRRP_Instance(VI_1) setting protocol
VIPs.
May 6 05:10:44 localhost Keepalived_healthcheckers: Netlink reflector reports IP
192.168.1.108 added
May 6 05:10:44 localhost Keepalived_vrrp: VRRP_Instance(VI_1) Sending gratuitous
ARPs on eth0 for 192.168.1.108
May 6 05:10:44 localhost Keepalived_vrrp: Netlink reflector reports IP
192.168.1.108 added
May 6 05:10:44 localhost avahi-daemon[2212]: Registering new address record for
192.168.1.108 on eth0.
May 6 05:10:49 localhost Keepalived_vrrp: VRRP_Instance(VI_1) Sending gratuitous
ARPs on eth0 for 192.168.1.108
```

很显然 VRRP 已经启动，还可以通过命令“ip addr”来检查主 Nginx 上的 IP 分配情况，通过下面的显示内容可以清楚地看到 VIP 地址已经绑定到主 Nginx 的机器上了。

```
ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:0c:29:51:59:df brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.103/24 brd 192.168.1.255 scope global eth0
    inet 192.168.1.108/32 scope global eth0
    inet6 fe80::20c:29ff:fe51:59df/64 scope link
        valid_lft forever preferred_lft forever
3: sit0: <NOARP> mtu 1480 qdisc noop
    link/sit 0.0.0.0 brd 0.0.0.0
```

在这个过程中，有大量的 VRRP 数据包，那么什么是 VRRP 呢？为了让大家对 Keepalived 有所了解，这里向大家详细说明下虚拟路由冗余协议（Virtual Router Redundancy Protocol, VRRP）。

随着 Internet 的迅猛发展，基于网络的应用逐渐增多，这就对网络的可靠性提出了越来越高的要求。斥资对所有的网络设备进行更新当然是一种很好的可靠性解决方案，但从保护现有资产的角度考虑，可以采用廉价冗余的思路，在可靠性和经济性方面找到平衡点。

虚拟路由冗余协议就是一种很好的解决方案。在该协议中，对共享多存取访问介质（如以太网）上的终端 IP 设备的默认网关（Default Gateway）进行冗余备份，当其中一台路由设备宕机时，备份路由设备及时接管转发工作，向用户提供透明的切换，提高了网络服务质量。

(1) 协议概述

在基于 TCP/IP 协议的网络中,为了保证没有直接物理连接的设备之间的通信,必须指定路由。目前常用的指定路由的方法有两种:一种是通过路由协议(比如:内部路由协议 RIP 和 OSPF)动态学习;另一种是静态配置。在每一个终端都运行动态路由协议是不现实的,大多数客户端操作系统平台都不支持动态路由协议,即使支持也会受到管理开销、收敛度、安全性等许多问题的限制。因此普遍采用对终端 IP 设备进行静态路由配置的方式,一般是给终端设备指定一个或多个默认网关。静态路由的方法简化了网络管理的复杂度,也减轻了终端设备的通信开销,但是它仍然有一个缺点:如果作为默认网关的路由器损坏,所有使用该网关为下一跳主机的通信必然要中断。即便配置了多个默认网关,如不重新启动终端设备,也不能切换到新的网关上。但是采用虚拟路由冗余协议(VRRP)就可以很好地避免静态指定网关的缺陷。

在 VRRP 协议中,有两组重要的概念:VRRP 路由器和虚拟路由器,主控路由器和备份路由器。VRRP 路由器是指运行 VRRP 的路由器,是物理实体,虚拟路由器是指 VRRP 协议创建的路由器,是逻辑概念。一组 VRRP 路由器协同工作,共同构成一台虚拟路由器。该虚拟路由器对外表现为一个具有唯一固定 IP 地址和 MAC 地址的逻辑路由器。处于同一个 VRRP 组中的路由器具有两种互斥的角色:主控路由器和备份路由器,一个 VRRP 组中有且只有一台处于主控角色的路由器,但可以有一个或多个处于备份角色的路由器。VRRP 协议使用选择策略从路由器组中选出一台作为主控路由器,负责 ARP 响应和转发 IP 数据包,组中的其他路由器作为备份的角色处于待命状态。当由于某种原因主控路由器发生故障时,备份路由器能在几秒钟的延时后升级为主控路由器。由于此切换非常迅速而且不用改变 IP 地址和 MAC 地址,故对终端使用者的系统来说是透明的。

(2) 工作原理

一个 VRRP 路由器有唯一的标识:VRID,范围为 0~255。该路由器对外表现为唯一的虚拟 MAC 地址,地址的格式为 00-00-5E-00-01-[VRID]。主控路由器负责对 ARP 请求用该 MAC 地址做应答。这样,无论如何切换,保证给终端设备的都是唯一一致的 IP 和 MAC 地址,减少了切换对终端设备的影响。

VRRP 控制报文只有一种:VRRP 通告(advertisement)。它使用 IP 多播数据包进行封装,组地址为 224.0.0.18,发布范围只限于同一局域网内。这点保证了 VRID 在不同的网络中可以重复使用。为了减少网络带宽的消耗,只有主控路由器才可以周期性地发送 VRRP 通告报文。若备份路由器在连续三个通告间隔内收不到 VRRP 或收到优先级为 0 的通告则启动新一轮的 VRRP 选举。

在 VRRP 路由器组中,按优先级选举主控路由器,VRRP 协议中的优先级范围是 0~255。若 VRRP 路由器的 IP 地址和虚拟路由器的接口 IP 地址相同,则称该虚拟路由器为 VRRP 组中的 IP 地址所有者,IP 地址所有者自动具有最高优先级 255。优先级 0 一般用在 IP 地址所有者主动放弃主控者角色时。可配置的优先级范围为 1~254。优先级的配置原则可以依据链路的速度、成本、路由器的性能和可靠性,以及其他管理策略来设定。在主控

路由器的选举中，高优先级的虚拟路由器会获胜，因此，如果在 VRRP 组中有 IP 地址所有者，则它总是作为主控路由的角色。对于有相同优先级的候选路由器，则按照 IP 地址的大小顺序进行选举。VRRP 还提供了优先级抢占策略，如果配置了该策略，高优先级的备份路由器便会剥夺当前低优先级的主控路由器而成为新的主控路由器。

为了保证 VRRP 协议的安全性，这里有两种安全认证措施：明文认证和 IP 头认证。明文认证方式要求：在加入一个 VRRP 路由器组时，必须同时提供相同的 VRID 和明文密码。这种方法可以避免在局域网内的配置错误，但不能防止通过网络监听方式获得密码。IP 头认证的方式提供了更高的安全性，能够防止报文重放和修改等攻击。

我们可以通过 TCPDump 抓包发现两台 Nginx 负载均衡器上均有大量 VRRP 包，在任何一台机器上开启 TCPDump 进行抓包，命令如下：

```
tcpdump vrrp
```

命令显示结果如下：

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
18:04:16.372116 IP 192.168.1.103 > VRRP.MCAST.NET: VRRPv2, Advertisement, vrid
51, prio 100, authtype simple, intvl 1s, length 20
18:04:17.374134 IP 192.168.1.103 > VRRP.MCAST.NET: VRRPv2, Advertisement, vrid
51, prio 100, authtype simple, intvl 1s, length 20
18:04:18.375461 IP 192.168.1.103 > VRRP.MCAST.NET: VRRPv2, Advertisement, vrid
51, prio 100, authtype simple, intvl 1s, length 20
18:04:19.376198 IP 192.168.1.103 > VRRP.MCAST.NET: VRRPv2, Advertisement, vrid
51, prio 100, authtype simple, intvl 1s, length 20
18:04:20.377229 IP 192.168.1.103 > VRRP.MCAST.NET: VRRPv2, Advertisement, vrid
51, prio 100, authtype simple, intvl 1s, length 20
18:04:20.378986 IP 192.168.1.104 > VRRP.MCAST.NET: VRRPv2, Advertisement, vrid
51, prio 99, authtype simple, intvl 1s, length 20
18:04:20.381515 IP 192.168.1.103 > VRRP.MCAST.NET: VRRPv2, Advertisement, vrid
51, prio 100, authtype simple, intvl 1s, length 20
18:04:21.383936 IP 192.168.1.103 > VRRP.MCAST.NET: VRRPv2, Advertisement, vrid
51, prio 100, authtype simple, intvl 1s, length 20
```

可以看到，优先级高的一方（即 priority 为 100 的机器）通过 VRRPv2 获得了 VIP 地址；而且 VRRPv2 包的发送极有规律，每秒发送一次，当然了，这是通过配置文件进行控制的。通俗地讲，它会不断地发送 VRRPv2 包来告诉从机（作为“老二”的机器），我是“老大”，VIP 地址我已经抢到手了，你不要再抢啦。

4. 用 nginx_pid.sh 脚本来监控 Nginx 进程

针对 Nginx+Keepalived 方案，编写的 Nginx 监控脚本 nginx_pid.sh 实现了真正意义上的高可用。此脚本的思路其实也很简单，即将其放置在后台一直监控 Nginx 进程。如果进程消失，则尝试重启 Nginx；如果失败，则立即停掉本机的 Keepalived 服务，让另一台负载均衡器接手。此脚本直接从生产环境下载，内容如下所示：

```
#!/bin/bash
while :
do
    nginxpid=`ps -C nginx --no-header | wc -l`
    if [ $nginxpid -eq 0 ];then
        /usr/local/nginx/sbin/nginx
        sleep 5
        if [ $nginxpid -eq 0 ];then
            /etc/init.d/keepalived stop
        fi
    fi
    sleep 5
done
```

然后将其置于后台运行“`sh /root/nginx_pid.sh &`”（也可以将此程序交由 Supervisor 托管），做到这步时大家要注意一下，这种写法是有问题的，我们用 root 用户退出终端后，此进程便会消失，正确写法为“`nohup /bin/bash /root/nginx_pid.sh &`”。

此脚本是直接生产服务器上下载的，大家不要怀疑它会引起死循环和有效性的问题。这是一个无限循环的脚本，放在主 Nginx 机器上（因为目前主要是由它来提供服务），每隔 5 秒执行一次，用 `ps -C` 命令来收集 Nginx 服务的 PID 值到底是否为 0，如果是 0 的话（即 Nginx 进程死掉了），尝试启动 Nginx 进程；如果继续为 0，即 Nginx 启动失败，则关闭本机的 Keepalived 进程，VIP 地址将会由备机接管。当然了，整个网站就会由备机的 Nginx 来提供服务了，这样就保证了 Nginx 进程的高可用（虽然在实际生产环境中基本没发现 Nginx 进程死掉的情况，多此一步操作算是有备无患吧）。我们在几次的线上维护工作中，手动人为重启了主 Nginx 服务器，而从 Nginx 在非常短的时间就切换过来了，客户没有因为网站故障而进行投诉，事实证明此脚本还是有效的。



说明 介绍一下 nohup 的作用。如果你正在运行一个进程，而且你觉得在退出账户时该进程还不应该结束，那么可以使用 nohup 命令，该命令可以在你退出 root 账户之后继续运行相应的进程，nohup 就是不挂起的意思（no hang up）。

5. 模拟故障测试

整套系统配置完成后，我们就可以通过 `http://192.168.1.108/` 来访问了，接下来要做一些模拟性的故障测试，比如关掉一台 Nginx 负载均衡器，抽掉某台 Web 机器的网线或直接关机，甚至停掉其中一台 Nginx 服务器的 Nginx 服务。我们会发现无论在什么情况下，Nginx+Keepalived 都可以正常提供服务，笔者的许多项目（包括电子商务网站）所用的都是此架构，并且已经在线上稳定运行好几年了。

6. Nginx 作为负载均衡器在工作中遇到的问题

（1）如何让 Nginx 负载均衡器也支持 HTTPS

要让 Nginx 支持 HTTPS，方法其实很简单，在负载均衡器上开启 SSL 功能，监听 443

端口(防火墙上也要做好映射),将证书放在 Nginx 负载均衡器上(而不是后面的 Web 服务器),即可轻松解决此问题,详见以下 nginx.conf 配置文件:

```
server
{
    listen 443;
    server_name www.cn7788.com;

    ssl on;
    ssl_certificate /usr/local/nginx/keys/www.cn7788.com.crt;
    ssl_certificate_key /usr/local/nginx/keys/www.cn7788.com.key;

    ssl_protocols SSLv3 TLSv1;
    ssl_ciphers ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:-LOW:-SSLv2:-EXP;
}
```

(2) 如何让后端的 Apache 服务器获取客户端的真实 IP

运行在后端 Apache 服务器上的应用所获取到的 IP 都是 Nginx 负载均衡所在服务器的 IP,或者是本机 127.0.0.1。查看 Apache 的访问日志,就会见到来来去去都是内网的 IP。虽然可以通过 Nginx 日志来判断客户端的 IP,但有些考虑不周全的应用,例如 Tattertools(一款优秀的博客程序)就会犯错,在后台的访问日志上总是显示访客数为 1,IP 来自 127.0.0.1。这时候就要想办法来处理了。其实我们可以通过修改 Nginx proxy 的参数令后端应用获取到 Nginx 发来的请求报文,并获取外网的 IP,在 Nginx 的配置文件中记得加上如下命令:

```
proxy_set_header    Host $host;
proxy_set_header    X-Real-IP $remote_addr;
proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
```

这仅仅是让 Nginx 获取外网 IP,Apache 未必买账呢!也就是说 Apache 端同样需要设置,搜寻了一下,发现 Apache 有一个来自第三方的 mod(模块)可配合 Nginx proxy 使用。下面简要说一下这个模块。

该模块相关说明可参考: <http://stderr.net/apache/rpaf/>。

该模块的下载地址为: <http://stderr.net/apache/rpaf/download/>。

该模块的最新版本是 mod_rpaf-0.6.tar.gz。

该模块的安装也相当简单,命令如下:

```
tar zxvf mod_rpaf-0.6.tar.gz
```

下载后解压,命令如下:

```
cd mod_rpaf-0.6
```

Apache 的目录可按自己的环境进行修改,现在选择相应的安装方式,命令如下:

```
/usr/local/apache/bin/apxs -i -c -n mod_rpaf-2.0.so mod_rpaf-2.0.c
```


完成后会在 `http.conf` 的 `LoadModule` 区域多加了一行内容，命令如下：

```
LoadModule mod_rpaf-2.0.so _module modules/mod_rpaf-2.0.so
```

经 Apache 2.2.6 的实验证明，使用这一行启动 Apache 的时候会报错。所以将其改为：

```
LoadModule rpaf_module modules/mod_rpaf-2.0.so
```

并在下方添加：

```
RPAFenable On
RPAFsethostname On
RPAFproxy_ips 127.0.0.1 192.168.1.101 192.168.102
RPAFheader X-Forwarded-For
```

在填写 Nginx 所在的内网 IP 时，Nginx 的内网地址是必写的，不然一样会失败，另外，有几个代理服务器的 IP 就写几个代理服务器的 IP。

保存退出后重启 Apache，再看看 Apache 的日志内容，应该已经很完美地解决了这个问题。

(3) 正确区分 Nginx 的分发请求

笔者在负责某个小项目时，原本是基于 Nginx 的 1+3 架构，开发人员突然要求增加一台机器（Windows Server 2003 系统），专门用于存放图片及 PDF 文件等，项目要求能在 Nginx 后的 3 台 Web 机器上显示图片，并且可以下载 PDF。当时笔者感觉有点为难，因为程序用的是 Zend Framework，所以一直在用正则表达式作为跳转。后来才想明白其中的“玄机”，IE 程序先在 Nginx 负载均衡器上提出申请，所以 `nginx.conf` 是在做分发而非正则跳转，此时最前端的 Nginx 既是负载均衡器也是反向代理，明白这个就好办了。另外要注意 `location /StockInfo` 与 `location ~ ^/StockInfo` 的差异性，Nginx 默认是正则优先的，顺便也说一下，`proxy_pass` 支持直接写 IP 的方式。Nginx 配置文件的部分代码如下所示：

```
upstream mysrv {
    ip_hash;
    server 192.168.110.62;
    server 192.168.110.63;
}

upstream myjpg {
    server 192.168.110.3:88;
}

server {
    listen 80;
    server_name web.tfzq.com;
    proxy_redirect off;

    location ~ ^/StockInfo{
        proxy_pass http://myjpg;
    }
}
```


关于这个案例，在此总结一下：目前此套架构在 2000 并发的电子商务网站上运行非常稳定，唯一不足之处就是 Nginx_backup 一直处于闲置状态。相对于双主 Nginx 负载均衡器而言，此架构比较简单，出问题的概率相对也较小，而且出问题时很容易排除故障，在网站收录方面需要考虑的问题也非常少，所以我一直采用这种方案。通过线上的观察，我们不难发现，Nginx 作负载均衡器 / 反向代理也是相当稳定的，可以媲美硬件级的 F5 了，相信这也是越来越多的朋友喜欢它的原因之一。Nginx+Keepalived 用于生产环境的优势还是很多的，不过由于其自身的限制，它目前只应用于 Web 集群环境；如果大家的网站或项目有这种需求，不妨考虑应用一下。

7.4.3 案例分享三：Nginx 主主负载均衡架构

在和一些朋友交流 Nginx+Keepalived 网站架构技术时，笔者虽然已多次成功实施 Nginx+Keepalived 项目方案，但这些都是在用单主 Nginx 工作，从 Nginx 长期只是处于备份状态，如果想让两台 Nginx 负载均衡器都处于工作状态，其实通过 Keepalived 很容易就可以实现。

一般为了维护方便，企业网站的服务器都在自己的内部机房里，只开放了 Keepalived 的 VIP 地址的两个端口 80、443，通过 Juniper SSG550 防火墙映射出去，外网 DNS 对应映射后的公网 IP。此架构的防火墙及网络安全说明如下。

此系统架构仅映射内网 VIP 的 80 及 443 端口于外网的 Juniper SSG550 防火墙下，其他端口均关闭，内网所有机器均关闭 iptables 防火墙；外网 DNS 指向通过 Juniper SSG550 映射出来的外网地址。

Nginx 主主架构服务器 IP 分配表如表 7-3 所示。

表 7-3 Nginx 主主架构服务器 IP 分配表

服务器名称	IP	用 途
Nginx-Master-1	192.168.1.1.5	提供负载均衡
Nginx-Master-2	192.168.1.1.6	提供负载均衡
Web1 服务器	192.168.1.1.7	提供 Web 服务
Web2 服务器	192.168.1.1.8	提供 Web 服务
VIP-1	192.168.1.8	集群 VIP 地址一
VIP-2	192.168.1.9	集群 VIP 地址二

Nginx 和 Keepalived 的安装比较简单，大家可以参考案例二，这里略过。以下附上两台 Nginx 负载均衡机器的 nginx.conf 配置文件，内容如下：

```
ser www www;
worker_processes 8;
pid /usr/local/nginx/logs/nginx.pid;
worker_rlimit_nofile 51200;
```

```

events
{
    use epoll;
    worker_connections 51200;
}

http
{
    include mime.types;
    default_type application/octet-stream;
    server_names_hash_bucket_size 128;
    client_header_buffer_size 32k;
    large_client_header_buffers 4 32k;
    client_max_body_size 8m;
    sendfile on;
    tcp_nopush on;
    keepalive_timeout 60;
    tcp_nodelay on;
    fastcgi_connect_timeout 300;
    fastcgi_send_timeout 300;
    fastcgi_read_timeout 300;
    fastcgi_buffer_size 64k;
    fastcgi_buffers 4 64k;
    fastcgi_busy_buffers_size 128k;
    fastcgi_temp_file_write_size 128k;
    gzip on;
    gzip_min_length 1k;
    gzip_buffers 4 16k;
    gzip_http_version 1.0;
    gzip_comp_level 2;
    gzip_types text/plain application/x-javascript text/css application/xml;
    gzip_vary on;

    upstream backend
    {
        server 192.168.1.17:80;
        server 192.168.1.18:80;
    }

    server
    {
        listen 80;
        server_name www.1paituan.com;
        location / {
            root /var/www/html;
            index index.php index.htm index.html;
            proxy_redirect off;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_pass http://backend;
        }
    }
}

```

```

location /nginx {
    access_log off;
    auth_basic "NginxStatus";
}

log_format access '$remote_addr - $remote_user [$time_local] "$request"
'$status $body_bytes_sent "$http_referer" ' '"$http_user_agent"
$http_x_forwarded_for';
access_log /data/logs/access.log access;
}
}

```

这里简单解释一下配置 Keepalived 文件的原理，其实就是通过 Keepalived 生成两个实例，两台 Nginx 互为备份，即第一台是第二台机器的备机，第二台机器也是第一台的备机，生成的两个 VIP 地址分别对应我们的网站 <http://www.lpaituan.com>，这样大家在公网上都可以通过 DNS 轮询来访问得到该网站。任何一台 Nginx 机器如果发生硬件损坏，Keepalived 会自动将它的 VIP 地址切换到另一台机器，而不影响客户端的访问，这个跟 LVS+Keepalived 多实例的原理是一样的，相信大家很容易就能明白。

主 Nginx 机器之一的 keepalived.conf 配置文件如下：

```

! Configuration File for keepalived
global_defs {
    notification_email {
        yuhongchun027@163.com
    }
    notification_email_from keepalived@chtopnet.com
    smtp_server 127.0.0.1
    smtp_connect_timeout 30
    router_id LVS_DEVEL
}

```

```

vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass lpaituan.com
    }
    virtual_ipaddress {
        192.168.1.8
    }
}

```

```

vrrp_instance VI_2 {
    state BACKUP
    interface eth0
    virtual_router_id 52
}

```

```

priority 99
advert_int 1
authentication {
    auth_type PASS
    auth_pass lpaituan.com
}
virtual_ipaddress {
    192.168.1.9
}
}

```

另一台 Nginx 机器的 keepalived.conf 配置文件如下:

```

! Configuration File for keepalived
global_defs {
    notification_email {
        yuhongchun027@163.com
    }
    notification_email_from keepalived@chtopnet.com
    smtp_server 127.0.0.1
    smtp_connect_timeout 30
    router_id LVS_DEVEL
}

```

```

vrrp_instance VI_1 {
    state BACKUP
    interface eth0
    virtual_router_id 51
    priority 99
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass lpaituan
    }
    virtual_ipaddress {
        192.168.1.8
    }
}

```

```

vrrp_instance VI_2 {
    state MASTER
    interface eth0
    virtual_router_id 52
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass lpaituan
    }
    virtual_ipaddress {

```

```
192.168.1.9
```

注意：这两台 Nginx 应该是互为主备的关系，所以在写 `keepalived.conf` 配置文件时，一定要注意 MASTER 和 BACKUP 的关系，并且还要注意各自的优先级，不是太熟悉的朋友建议直接用以上的脚本来操作。如果此步遇到了问题，建议用 TCPDump 抓包分析排障，正常情况下它们都应该各自向对方发送 VRRP 包。

大家都知道 Keepalived 是实现不了程序级别的高可用的，所以我们要写 Shell 脚本来实现 Nginx 的高可用，脚本 `/root/nginx_pid.sh` 内容如下：

```
#!/bin/bash
while :
do
    nginxpid=`ps -C nginx --no-header | wc -l`
    if [ $nginxpid -eq 0 ];then
        /usr/local/nginx/sbin/nginx
        sleep 5
        if [ $nginxpid -eq 0 ];then
            /etc/init.d/keepalived stop
        fi
    fi
    sleep 5
done
```

下面分别在两台主 Nginx 上执行如下命令：

```
nohup sh /root/nginxpid.sh &
```

正常启动两台主 Nginx 的 Nginx 和 Keepalived 程序后，这两台机器的正常 IP 显示应该如下。

IP 为 192.168.1.5 的机器的 `ip addr` 命令显示结果为：

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
link/ether 00:0c:29:99:fb:32 brd ff:ff:ff:ff:ff:ff
inet 192.168.1.5/24 brd 192.168.1.255 scope global eth0
inet 192.168.1.8/32 scope global eth0
```

IP 为 192.168.1.6 的机器的 `ip addr` 命令显示结果为：

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
```



```

link/ether 00:0c:29:7d:58:5e brd ff:ff:ff:ff:ff:ff
inet 192.168.1.6/24 brd 192.168.1.255 scope global eth0
inet 192.168.1.9/32 scope global eth0
inet6 fe80::20c:29ff:fe7d:585e/64 scope link
valid_lft forever preferred_lft forever
3: sit0: <NOARP> mtu 1480 qdisc noop
   link/sit 0.0.0.0 brd 0.0.0.0

```

测试过程如下：

- 1) 分别在这两台主 Nginx 上用 killall 杀掉 Nginx 进程，然后在客户端分别访问 192.168.1.8 和 192.168.1.9 这两个 IP（模拟 DNS 轮询）看能否正常访问 Web 服务器。
- 2) 尝试重启 192.168.1.5 的主 Nginx 负载均衡器，测试过程如上。
- 3) 尝试重启 192.168.1.6 的主 Nginx 负载均衡器，测试过程如上。
- 4) 尝试分别关闭 192.168.1.5 和 192.168.1.6 的机器，测试过程如上，以确定是否影响网站的正常访问。

目前投入生产要解决的问题为：

- ☐ 日志收集系统要重新部署，现在访问日志是分布在两台 Nginx 负载均衡器上的。
- ☐ 要考虑谷歌和百度域名收录的问题（注意看对网站的 SEO 有无影响）。
- ☐ 要考虑 DNS 轮询的问题。
- ☐ 证书的问题，两台 Nginx 负载均衡机器都需要部署 GeoTrust 证书。
- ☐ Nginx 主主架构较之 Nginx 主备架构维护起来更为复杂，这点希望大家在工作中注意。
- ☐ 实施此网站的架构时不要影响网站的业务系统，事实上很多业务系统在程序上都做了防作弊措施，所以在实施时考虑问题一定要全面。

7.4.4 案例分享四：生产环境下的高可用 NFS 文件服务器

某日例行检查机房的服务器时，发现笔者的 DRBD+Heartbeat+NFS 文件服务器已经稳定运行了 1061 天（两台 CentOS 5.3 i386，DELL2950 机器，用作图片及代码存放文件服务器），稳定性相当不错，鉴于 DRBD 已作为 MySQL 官方推荐的用于实现 MySQL 高可用的一种非常重要的方式，建议大家掌握这个知识点。在介绍过程中，如果有需要注意的地方会重点说明，整个测试过程均参考了线上文档。

分布式复制块设备（Distributed Replicated Block Device，DRBD）是一种基于 Linux 的软件组件，它是由内核模块和相关程序组成的，可通过网络镜像促进共享存储系统的替换。也就是说：当你将数据写入本地 DRBD 设备上的文件系统时，数据会同时被发送到网络中的另外一台主机之上，并以完全相同的形式记录在一个文件系统中（实际上文件系统的创建也是由 DRBD 的同步来实现的）。本地节点（主机）与远程节点（主机）的数据可以保证实时同步，并且可以保证 I/O 的一致性。所以当本地节点的主机出现故障时，远程节点的主机上还保留着一份完全相同的数据，可以继续使用，以达到高可用的目的。

DRBD 的工作原理如图 7-8 所示。

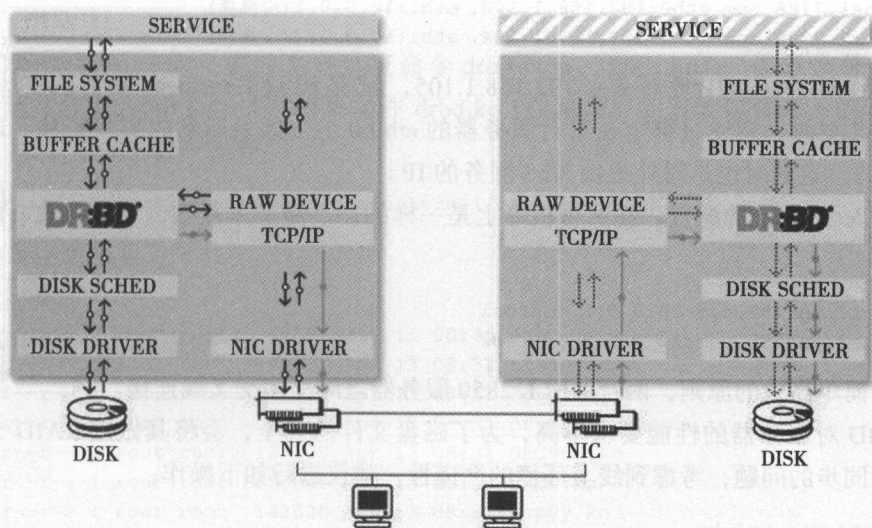


图 7-8 DRBD 工作原理图

我们可以这样理解 DRBD，它其实就是网络 RAID-1，两台服务器中就算其中的某台因电源或主板损坏而宕机也不会对数据有任何影响（可以用硬盘 RAID-1 来理解 DRBD），而真正的热切换可以通过 Heartbeat 来实现，这样的切换过程跟 Keepalived 类似，非常短且不需要人为干预。

DRBD 需要构建在底层设备之上，而且还需要构建出一个块设备来。对于用户来说，一个 DRBD 设备就像是一块物理磁盘，可以在磁盘内创建文件系统。DRBD 所支持的底层设备如下：

- ❑ 单独的磁盘。
- ❑ 磁盘的某一个分区。
- ❑ 一个 soft RAID 设备。
- ❑ 一个 LVM 的逻辑卷。
- ❑ 一个 EVMS (Enterprise Volume Management System, 企业卷管理系统) 的卷。
- ❑ 其他任何的块设备。

此项目中采用的是第一种，即用单独的磁盘来做 DRBD。

下面将介绍 DRBD+Heartbeat+NFS 的详细配置过程。

1. 做好整个环境的准备工作

两台服务器均要提前做好准备工作，比如设置好 hosts 文件并进行 NTP 对时。

primary 服务器: centos1.cn7788.com, 单独拿一块硬盘 sdb 作为 DRBD

secondary 服务器: centos2.cn7788.com, 单独拿一块硬盘 sdb 作为 DRBD

网络拓扑很简单，命令如下：

```
centos1.7788.com eth0:192.168.1.103, eth1:10.0.0.1(心跳线)
centos2.7788.com eth0:192.168.1.104, eth1:10.0.0.2(心跳线)
```

DRBD 对外的 VIP 地址是：192.168.1.105，这是通过 Heartbeat 来实现的，原理跟 Keepalived 类似，它通过绑定在某台服务器的 eth0:0 上，如果遇到故障就转移，以达到高 HA 的目的。这同时也是对外提供 NFS 服务的 IP。

关于 /etc/hosts 的配置，在两台机器上是一样的，不需要太复杂，只须配置心跳部分即可，即：

```
centos1.cn7788.com 10.0.0.1 centos1
centos2.cn7788.com 10.0.0.2 centos2
```

本着简单高效的原则，两台 DELL 2850 服务器之间采用交叉线连接。

DRBD 对服务器的性能要求不高，为了磁盘文件的安全，会将其做成 RAID 5。另外，关于时间同步的问题，考虑到线上环境的严谨性，建议进行如下操作：

```
ntpdate ntp.api.bz
```

最后，如果想让主机名的名字永久生效，建议重启机器，核对检查所有配置是否生效；最重要的心跳线一定要保持 ping 通的状态，我们在工作后期对 IP 进行了迁徙工作，发现只要不人为拔掉心跳线，基本上不会发生“脑裂”现象。另外，记得关掉两台机器的 iptables 和 SELinux，以免影响结果。

2.DRBD 的安装和配置过程

1) 首先安装依赖库，以下进行安装与测试的均为 CentOS 5.3 i386 系统，这里采用的软件安装方式主要以源码为主，yum 的安装方法可以参考后面的章节，命令如下：

```
yum install gcc gcc-c++ make glibc flex
```

2) 在 primary 和 secondary 上都要使用相同的安装方法，如下所示：

```
cd /usr/local/src
wget http://oss.linbit.com/drbd/8.0/drbd-8.0.6.tar.gz
tar zxvf drbd-8.0.6.tar.gz
cd drbd-8.0.6
make
```

这里需要说明一下的是，如果没有更改内核则可以直接运行 make，程序会直接到 /lib/module 里面去寻找系统环境，如果是新的内核则需要指定内核所在的位置，如 make DKDIR=/usr/src/linux 或 make DKDIR=/usr/src/kernel/，这个目录就是系统内核所在的位置，否则 make 的时候会报“Could not determine uts_release”的错误而中断掉。因为前面在进行 LVS 等实验演示时升级了内核，所以在这里也需要指定内核位置，笔者用的是 make DKDIR=/usr/src/kernel/。如果大家在安装时遇到报错问题，以上方法都可以尝试一下；如

果是新内核升级，还要记得重启一下机器，不然也会安装不成功的。命令如下：

```
make install
```

3) DRBD 程序安装完成后主要生成命令 `drbdsetup`、`drbdadmin`，配置文件 `/etc/drbd.conf`、启动文件 `/etc/init.d/drbd` 及模块文件 `drbd.ko`（在编译好的安装包目录下的 `drbd` 中可以找到），可以用如下命令查看：

```
ll /lib/modules/2.6.18-238.9.1.el5/kernel/drivers/block/
```

命令显示结果如下所示：

```
total 3652
drwxr-xr-x 2 root root      4096 May 15 00:45 aoe
-rwxr--r-- 1 root root    141952 Apr 13 08:31 cciss.ko
-rwxr--r-- 1 root root     71712 Apr 13 08:31 cpqarray.ko
-rwxr--r-- 1 root root     39264 Apr 13 08:31 cryptoloop.ko
-rwxr--r-- 1 root root    128520 Apr 13 08:31 DAC960.ko
-rw-r--r-- 1 root root   2858045 May 15 00:54 drbd.ko
-rwxr--r-- 1 root root    142800 Apr 13 08:31 floppy.ko
-rwxr--r-- 1 root root     56824 Apr 13 08:31 loop.ko
-rwxr--r-- 1 root root     51984 Apr 13 08:31 nbd.ko
drwxr-xr-x 2 root root      4096 May 15 00:45 paride
-rwxr--r-- 1 root root     76216 Apr 13 08:31 pktcdvd.ko
-rwxr--r-- 1 root root     61744 Apr 13 08:31 sx8.ko
-rwxr--r-- 1 root root     45648 Apr 13 08:31 virtio_blk.ko
```

所有命令和配置文件都可以在源码包编译成功的目录下找到。

4) DRBD 采用的是模块控制的方式，所以要先加载 `drbd.ko` 模块，命令如下：

```
modprobe drbd
```

查看 DRBD 模块是否已经加载到内核中了（有的话表示已经成功加载到内核中了），命令如下：

```
lsmod | grep drbd
```

显示结果如下所示：

```
drbd      226608  0
```

5) 确认两台要镜像的机器运行是否正常，它们之间的网络是否通畅，需要加载的硬盘是否处于 `umount` 状态，命令略过。

6) 在两台主机上都创建硬件设备 DRBD，命令如下：

```
mknod /dev/drbd0 b 147 0
```

因为两台主机都只有一块 DRBD 设备，所以这里不需要再创建多个了。

7) 两台机器将 `/dev/sdb1` 互为镜像（两台机器的配置相同），这里只列出 `centos1` 的操作步骤，`centos2` 的操作步骤可依次类推，具体如下：


```
yum -y install portmap
yum -y install nfs
mkdir /drbd
```

创建共享目录，用 vim 编辑 /etc/exports 文件，内容如下：

```
/d 10.1.2.0/255.255.252.0 (rw,no_root_squash,no_all_squash,sync)
```

这里只允许 10.1.0.0 网段的机器共享，也是基于安全的考虑。

然后配置 NFS 服务涉及的 portmap 和 nfs 服务，命令如下：

```
service portmap start
chkconfig --level 3 portmap on
chkconfig --level 3 nfs off
```

NFS 服务不需要启动，也不需要设置成开机自动运行，这些都将由后面的 Heartbeat 来完成。

8) 配置 DRBD。DRBD 运行时，会读取一个配置文件 /etc/drbd.conf，这个文件里描述了 DRBD 设备与硬盘分区的映射关系，以及 DRBD 的一些配置参数，编辑 /etc/drbd.conf 文件，内容如下：

```
resource r0 {
    protocol C;

    startup { wfc-timeout 0; degr-wfc-timeout 120; }
    disk { on-io-error detach; }
    net {
        timeout 60;
        connect-int 10;
        ping-int 10;
        max-buffers 2048;
        max-epoch-size 2048;
    }
    syncer { rate 30M; }

    on centos1.7788.com {
        device /dev/drbd0;
        disk /dev/sdb;
        address 10.0.0.1:7788;
        meta-disk internal;
    }
    on centos2.7788.com {
        device /dev/drbd0;
        disk /dev/sdb;
        address 10.0.0.2:7788;
        meta-disk internal;
    }
}
```

其中：

- ❑ resource r0 表示创建的资源名字。
- ❑ syncer C 表示采用 C 协议，如果收到远程主机的写入确认，则认为写入完成。syncer 选项是设备主备节点同步时的网络速率最大值。每个主机的说明都以 “on” 开头，然后是各自的主机名，再后面的 {} 里是这个主机的配置，监听端口为 7788。
- ❑ meta-disk internal 表示在同一个局域网内。

9) 启动 DRBD，激活前面配置的 DRBD 资源 r0 (两个节点都要执行)。在启动 DRBD 之前，需要在两台主机的 hdb1 分区上分别创建供 DRBD 记录信息的数据块，并在两台主机上分别执行如下命令：

```
drbdadm create-md r0
```

命令显示结果如下所示：

```
Valid meta-data already in place, recreate new?
[need to type 'yes' to confirm] yes
Creating meta data...
initialising activity log
NOT initialized bitmap (32 KB)
New drbd meta data block successfully created.
```

创建 r0 的资源，其中 r0 是我们在 drbd.conf 里定义的资源名称，最后一行显示创建 r0 资源成功。

现在可以启动 DRBD 了，分别在两台主机上执行，命令如下所示：

```
/etc/init.d/drbd start
```

设置 DRBD 开机时自动启动，命令如下：

```
chkconfig drbd on
```

现在可以查看 DRBD 当前的状态，在 centos1 上执行如下命令：

```
cat /proc/drbd
```

此命令执行后结果如下所示：

```
version: 8.0.0 (api:86/proto:86)
SVN Revision:22713 build by root@centos1, 2008-06-27 14:07:14
1: cs:Connected st:Secondary/Secondary ds:Inconsistent/Inconsistent C r—
ns:0 nr:0 dw:0 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0
resync: used:0/31 hits:0 misses:0 starving:0 dirty:0 changed:0 act_log:
      used:0/257 hits:0 misses:0 starving:0 dirty:0 changed:0
```

第一行的 “st” 表示两台主机的状态都是备机状态。

“ds” 是磁盘状态，显示其状态 “不一致”。这是因为 DRBD 无法判断哪一方为主机，应该以哪一方的磁盘数据作为标准数据。所以我们需要初始化一个主机，考虑放在 centos1 上执行。

10) 初始化 centos1 机器 (这步只要在主节点上操作即可)：

```
drbdsetup /dev/drbd0 primary -o
drbdadm primary r0
```

第一次设置主节点时用 `drbdadm` 命令会失败，所以先用 `drbdsetup` 来进行操作，以后就可以用 `drbdadm` 命令了。

再次查看 DRBD 当前的状态，命令如下：

```
cat /proc/drbd
```

显示结果如下：

```
version: 8.0.6(api:86/proto:86)
SVNRevision:2713build by root@centos1, 2008-06-27 14:07:14
1: cs:SyncSource st:Primary/Secondary ds:UpToDate/Inconsistent C r—
ns:18528 nr:0 dw:0 dr:18528 al:0 bm:1 lo:0 pe:0 ua:0 ap:0
[>.....] sync'ed: 0.3% (8170/8189)M
finish: 6:46:43 speed: 336 (324) K/sec
resync: used:0/31 hits:1156 misses:2 starving:0 dirty:0 changed:2
act_log: used:0/257 hits:0 misses:0 starving:0 dirty:0 changed:0
```

现在主备机的状态分别是“主/备”，主机磁盘状态是“实时”，备机状态是“不一致”。

在第 3 行中可以看到数据正在同步，即主机正在将磁盘上的数据传递到备机上，现在的进度是 0.3%。

设置完之后第一次同步的耗时会比较长，因为需要把整个分区的数据全部同步一遍。

第一次同步完成之后，就可以对 DRBD 的设备创建文件系统了。

稍等一段时间，在数据同步完成之后再查看一下两台机器的 DRBD 状态，命令如下：

```
[root@centos1 ~]# service drbd status
```

显示结果如下所示：

```
SVN Revision: 3048 build by root@centos1.7788.cn, 2010-01-20 06:09:12
0: cs:Connected st:Primary/Secondary ds:UpToDate/UpToDate C r—
```

查看 centos2 机器的 DRBD 状态，命令如下：

```
[root@centos2 ~]# service drbd status
```

显示结果如下所示：

```
SVN Revision: 3048 build by root@centos2.7788.cn, 2010-01-20 06:09:02
0: cs:Connected st:Secondary/Primary ds:UpToDate/UpToDate C r—
```

现在磁盘的状态都是“实时”的，表示数据同步完成了。

在查看 DRBD 的实时状态时，命令如下（两个命令任选一个即可）：


```
service drbd status
cat /proc/drbd
```

11) DRBD 的使用。现在可以把主机上的 DRBD 设备挂载到一个目录上使用了。备机的 DRBD 设备无法被挂载，因为它是用来接收主机数据的，由 DRBD 负责操作。

在 centos1 主服务器上执行如下命令：

```
mkfs.ext3 /dev/drbd0
mount /dev/drbd0 /drbd
```

现在，可以对 /drbd 分区进行读写操作了。

 **注意** secondary 节点上不允许对 DRBD 设备进行任何操作，包括只读。所有的读写操作只能在 primary 节点上进行，只有当 primary 节点挂掉时，secondary 节点才能提升成为 primary 节点，继续进行读写操作。

截止到这步，DRBD 的安装就算成功了。那么我们究竟应该如何保证 DRBD 机器的高可用呢？这就要用 Heartbeat 来实现了。另外，如果启动了 Heartbeat 服务，就不需要再手动 mount 了，Heartbeat 会自动 mount。

参考文档：

DRBD 的官方网站：<http://www.drbd.org/>。

源码下载地址：<http://oss.linbit.com/drbd>。

FAQ：<http://www.linux-ha.org/DRBD/FAQ>。

3. Heartbeat 的配置过程

1) 在两台主机上分别安装 Heartbeat，命令如下：

```
yum -y install heartbeat
```

奇怪的是，此命令要执行两次，不然 Heartbeat 安装不上去，这个也算是安装 Heartbeat 的一个 Bug 吧。

整个故障描述如下，很明显第一次安装 Heartbeat 包没有成功：

```
error: %pre(heartbeat-2.1.3-3.el5.centos.x86_64) scriptlet failed, exit status 9
error: install: %pre scriptlet failed (2), skipping heartbeat-2.1.3-3.el5.centos
Installed:
heartbeat.x86_64 0:2.1.3-3.el5.centos
Dependency Installed:
heartbeat-pils.x86_64 0:2.1.3-3.el5.centos
heartbeat-stonith.x86_64 0:2.1.3-3.el5.centos Complete!
```

本来要安装 4 个软件包，结果只安装成功了 3 个，所以必须要再执行一次如下命令：

```
yum -y install heartbeat
```

其中 Heartbeat 配置共涉及 4 个文件：

```
/etc/ha.d/ha.cf
/etc/ha.d/haresources
/etc/ha.d/authkeys
/etc/ha.d/resource.d/killlnfsd
```

2) 对两个节点机器进行配置，配置文件都是一样的，文件内容如下：

```
logfile /var/log/ha-log #定义HA的日志名字及存放位置
logfacility local0
keepalive 2 #设定心跳（监测）时间为2秒
deadtime 5 #死亡时间定义为5秒
ucast eth1?10.0.0.2 #采用单播方式，IP地址指定为对方IP
auto_failback off #服务器正常后由主服务器接管资源，另一台服务器放弃该资源
node centos1.7788.com??centos2.7788.com #定义节点
```

3) 编辑双机互连验证文件 /etc/ha.d/authkeys，直接用 vim 来编辑，命令如下所示：

```
auth 1
1 crc
```

需要将 /etc/ha.d/authkeys 设为 600 的权限，命令如下：

```
chmod 600 /etc/ha.d/authkeys
```

4) 编辑集群资源文件 /etc/haresources，内容如下所示：

```
centos1.cn7788.com IPaddr::192.168.1.108/24/eth0 drbdisk::r0 Filesystem::/dev/
drbd0::/drbd::ext3 killnfsd
```



注意 此文件在两台机器上的配置是一样的，强烈建议不要将另一台机器配置成 centos2.7788.com，最好的做法是先只在 centos1 机器上进行配置，然后将配置文件 scp 或 rsync 到 centos2 机器上面。

5) 编辑脚本文件 killnfsd，目的其实就是重启 NFS 服务。这是因为 NFS 服务切换后，必须要重新 mount 一下 NFS 服务共享出来的目录，否则会出现 “stale NFS file handle” 的错误，我们直接用 vim 来进行编辑，命令如下所示：

```
vim /etc/ha.d/resource.d/killnfsd
```

脚本内容如下所示：

```
killall -9 nfsd; /etc/init.d/nfs restart; exit 0
```

给予 killnfsd 执行的权限，命令如下所示：

```
[root@centos1 ha.d]# chmod 755 /etc/ha.d/resource.d/killnfsd
```

6) 在两个节点上启动 Heartbeat，可以先在主节点启动，命令如下所示：

```
[root@centos1 /]# service heartbeat start
[root@centos2 /]# service heartbeat start
```

这时就可以在另外的机器上面正常挂载 192.168.1.106:/drbd/ 到自己的 /mnt/data 下进行读写了，客户端会认为这仅仅是一个提供 NFS 的机器。192.168.1.108 是 Heartbeat 产生的 VIP 地址，也是对外提供 NFS 服务的地址。

4. 进行数据测试工作和故障模拟

虽然笔者的线上环境已经很稳定了，但为了让大家熟悉 DRBD+Heartbeat 的模式，还是

按步骤进行了相关的操作，建议完成如下测试后再查看 Heartbeat 是否能做到真正的热切换。

1) 在另外的某一台 Linux 机器上挂载 192.168.1.108:/drbd，在向里面写数据时，忽然重新启动主 DRBD，看此时写数据是否受到了影响，你可能会发现 DRBD+Heartbeat 正常切换还是需要些时间的。

2) 正常状态下将 Primary 关机，然后查看数据有无问题，观察 DRBD 的 status；等主机启动后，再观察变化，然后再将 Secondary 关机，再启动，观察 DRBD 的变化，以及 Heartbeat 是否起了作用。

3) 假设此时停用了 Primary 的 eth0 网卡，然后直接在 Secondary 上进行主 Primary 主机的提升，并且也给 mount 了，你可能会发现在 Primary 上测试拷入的文件确实同步过来了。之后把 Primary 的 eth0 网卡恢复后，看看有没有自动恢复主从关系，经过支持查询，你可能会发现 DRBD 检测出现了“脑裂”的状况，两个节点各自都成单点了，故障描述如下：“Split-Brain detected, dropping connection!”这就是传说中的“脑裂”，DRBD 官方推荐手动恢复（生产环境下出现这个问题的概率很低，谁会故意去触动生产中的服务器网线呢？谁又会像这样故意去折腾线上的生产服务器呢？）。我们在工作中进行 IP 迁徙时，会发现只要不触动心跳线，产生“脑裂”情况的概率真的很低。

以下为手动解决“脑裂”问题的方法。

a) 在 Secondary 主机上执行如下命令：

```
drbdadm secondary r0
drbdadm disconnect all
drbdadmin -discard-my-data connect r0
```

b) 在 Primary 主机上执行如下命令：

```
drbdadm disconnect all
drbdadm connect r0
```

4) 假设 Primary 主机因为硬件损坏，需要将 Secondary 提升成 Primary 主机，应该如何处理呢？方法如下。

a) 先在 Primary 主机上卸掉 DRBD 设备，命令如下：

```
umount /d
```

然后将主机降为备机，命令如下：

```
[root@centos1 /]# drbdadm secondary r0
```

观察其状态，命令如下：

```
[root@centos1 /]# cat /proc/drbd
```

显示结果如下所示：

```
1: cs:Connected st:Secondary/Secondary ds:UpToDate/UpToDate C r—
```


现在，两台主机都是备机了，我们可以通过 `st` 命令观察得到此信息。

b) 将 centos2 备机升级为 Primary 主机，命令如下所示：

```
[root@centos2 /]# drbdadm primary r0
[root@centos2 /]# cat /proc/drbd
1: cs:Connected st:Primary/Secondary ds:UpToDate/UpToDate C r—
```

现在备机就成功转为主机了。由于 centos2 上面也有完整的数据，所以整个切换过程不会发生数据丢失的现象，保证了数据的高可用性。

DRBD+Heartbeat+NFS 高可用文件服务器的整个构建过程还是比较复杂的，这里有个建议：先在局域网或 VMware 环境下将整个过程多测试几次，这样大家自然而然就会清楚其中的逻辑顺序了。大家也可以像笔者一样将这些操作步骤形成工作文档，这样，在线上环境中实施时会非常容易，逐步照做就行了。DRBD 也是现在很流行的技术，很多互联网公司将其作为高可用文件服务器方案，不仅如此，DRBD+Heartbeat 也适合做生产环境下的高可用 MySQL 服务（这个也是 MySQL 官方推荐的），所以推荐大家掌握 DRBD+Heartbeat+NFS 的配置方法。

7.4.5 案例分享五：生产环境下的 MySQL DRBD 双机高可用

鉴于 DRBD 的稳定性，笔者针对电子商务网站采用的也是这种 MySQL 高可用方案。由于系统安装初始化工作并非是本人进行的（DELL 供应商在机器进机房上架之前就已将系统安装好并且是最小化安装），在安装完 DRBD 包后“`modprobe drbd`”报错，在加载 DRBD 模块时经常报错，报错信息如下所示：“`FATAL: Module drbd not found drbd`”，后面发现是系统安装了双内核并使用新内核启动的原因，所以执行命令 `default=1`，回退到老版本内核运行，此报错就没有了，`/etc/grub.conf` 配置文件如下所示：

```
default=1
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title CentOS (2.6.18-238.el5xen)
    root (hd0,0)
    kernel /xen.gz-2.6.18-238.el5
    module /vmlinuz-2.6.18-238.el5xen ro root=LABEL=/ rhgb quiet
    module /initrd-2.6.18-238.el5xen.img
title CentOS-base (2.6.18-238.el5)
    root (hd0,0)
    kernel /vmlinuz-2.6.18-238.el5 ro root=LABEL=/ rhgb quiet
    initrd /initrd-2.6.18-238.el5.img
```

服务器的型号是 DELL R710（双至强 Xeon E5606 CPU），系统采用的 CentOS 5.8 x86_64，由 6 块 SAS 600G 硬盘作为 RAID 10，考虑到 RAID 10 作为文件系统的高效及速度，这里单独划分了接近 1.5TB 的硬盘空间给 DRBD 系统使用（在安装系统时选择将此空间作

为空闲空间,此数据库主要是为订单系统提供服务,所以不用考虑多维度数据增长的问题,基本上5~8年之内是可以满足需求的),此外还采用了两根交叉线作为心跳线,思科 CISCO?WS-C2960S-24TS-L 作为交换机设备。

两台机器的基本情况如下所示:

```
centos1.mypharma.com 112.112.68.170, 心跳线为: 192.168.1.1 10.0.0.1
centos2.mypharma.com 112.112.68.172, 心跳线为: 192.168.1.2 10.0.0.2
Heartbeat的vip为 112.112.68.174
```

两台机器的 hosts 文件内容分别如下:

```
112.112.68.170 centos1.mypharma.com centos1
112.112.68.172 centos2.mypharma.com centos2
```

两台机器的 hostname 主机名及 NTP 对时应应在实验前就配置好,并关闭 iptables 和 SELinux,硬盘情况可用 fdisk 来查看,命令如下:

```
fdisk -l
```

此命令显示结果如下所示:

```
Disk /dev/sda: 1798.6 GB, 1798651772928 bytes
255 heads, 63 sectors/track, 218673 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/sda1	*	1	16	128488+	83	Linux
/dev/sda2		17	5237	41937682+	82	Linux swap / Solaris
/dev/sda3		5238	11764	52428127+	83	Linux
/dev/sda4		11765	218673	1661996542+	5	Extended
/dev/sda5		11765	15710	31696213+	83	Linux
/dev/sda6		15711	198076	1464854863+	83	Linux

1. DRBD 的部署安装

两台机器分别用如下命令来安装 DRBD 软件:

```
yum -y install drbd83 kmod-drbd83
```

载入 DRBD 模块并检测模块载入是否正常,命令如下:

```
modprobe drbd
lsmod | grep drbd
```

如果正确显示了类似如下的信息,则表示 DRBD 已成功安装:

```
drbd      300440  4
```

用如下命令查看两台机器的 drbd.conf 配置文件,内容如下(两台机器的配置是一样的):

```
global {
```

```

# minor-count dialog-refresh disable-ip-verification
usage-count no;          #统计DRBD的使用次数
}
common {
    syncer { rate 30M; } #同步速率，视带宽而定
}
resource r0 {             #创建一个资源，名字叫"r0"
    protocol C;           #选择的是DRBD的C协议（数据同步协议，C为收到数据并写入后返回，确认成功）
    handlers {            #默认DRBD的库文件
        pri-on-incon-degr "/usr/lib/drbd/notify-pri-on-incon-degr.sh; /usr/lib/drbd/
            notify-emergency-reboot.sh; echo b > /proc/sysrq-trigger ; reboot -f";
        pri-lost-after-sb "/usr/lib/drbd/notify-pri-lost-after-sb.sh; /usr/lib/drbd/
            notify-emergency-reboot.sh; echo b > /proc/sysrq-trigger ; reboot -f";
        local-io-error "/usr/lib/drbd/notify-io-error.sh;
            /usr/lib/drbd/notify-emergency-shutdown.sh; echo o > /proc/sysrq-trigger ; halt -f";
        # fence-peer "/usr/lib/drbd/crm-fence-peer.sh";
        # split-brain "/usr/lib/drbd/notify-split-brain.sh root";
        # out-of-sync "/usr/lib/drbd/notify-out-of-sync.sh root";
        # before-resync-target "/usr/lib/drbd/snapshot-resync-target-lvm.sh -p 15 -- -c 16k";
        # after-resync-target /usr/lib/drbd/unsnapshot-resync-target-lvm.sh;
    }
    startup {
        # wfc-timeout degr-wfc-timeout outdated-wfc-timeout wait-after-sb
        wfc-timeout 120;
        degr-wfc-timeout 120;
    }
    disk {
        # on-io-error fencing use-bmbv no-disk-barrier no-disk-flushes
        # no-disk-drain no-md-flushes max-bio-bvecs
        on-io-error detach;
    }
    net {
        # sndbuf-size rcvbuf-size timeout connect-int ping-int ping-timeout max-buffers
        # max-epoch-size ko-count allow-two-primaries cram-hmac-alg shared-secret
        # after-sb-0pri after-sb-1pri after-sb-2pri data-integrity-alg no-tcp-cork
        max-buffers 2048;
        cram-hmac-alg "sha1";
        shared-secret "123456";      #DRBD同步时使用的验证方式和密码信息
        #allow-two-primaries;
    }
    syncer {
        rate 30M;
        # rate after al-extents use-rle cpu-mask verify-alg csums-alg
    }
    on centos1.mypharma.com {      #设定一个节点，分别以各自的主机名命名
        device /dev/drbd0;         #设定资源设备/dev/drbd0 指向实际的物理分区 /dev/sda6
        disk /dev/sda6;
        address 192.168.1.1:7788;  #设定监听地址及端口
        meta-disk internal;
    }
    on centos2.mypharma.com {      #设定一个节点，分别以各自的主机名命名

```

```

device /dev/drbd0;          #设定资源设备/dev/drbd0 指向实际的物理分区 /dev/sdb1
disk /dev/sda6;
address 192.168.1.2:7788;    #设定监听地址及端口
meta-disk internal;        #表示是在同一个局域网内
}
}

```

然后执行 `drbdadm create-md r0` 命令来创建 DRBD 元数据信息，两台机器都需要执行此命令，如果执行命令后出现如下报错：

```

Device '0' is configured!
Command 'drbdmeta 0 v08 /dev/sda6 internal create-md' terminated with exit code 20
drbdadm create-md r0: exited with code 20

```

那么为了修复这个错误，建议用 `dd` 破坏文件分区（注意，磁盘原有的数据会被全部清除掉，因此希望谨慎操作），命令如下：

```
dd if=/dev/zero of=/dev/sda6 bs=1M count=100
```

之后，在 `centos1` 的机器上执行如下命令：

```
[root@centos1 ~]# drbdadm create-md r0
```

命令显示结果如下：

```

Writing meta data...
initializing activity log
NOT initialized bitmap
New drbd meta data block successfully created.

```

在 `centos2` 的机器上执行如下命令：

```
[root@centos2 ~]# drbdadm create-md r0
```

命令显示结果如下：

```

Writing meta data...
initializing activity log
NOT initialized bitmap
New drbd meta data block successfully created.

```

若两台机器都有上面的显示结果则表明一切都是正常的。

现在启动 DRBD 设备，在两台机器上分别执行如下命令：

```
service drbd start
```

如果此时没有正常关闭 `iptables` 服务，则会产生报错信息，报错信息如下：

```

Starting DRBD resources: [ d(r0) s(r0) n(r0) ].....
*****
DRBD's startup script waits for the peer node(s) to appear.
- In case this node was already a degraded cluster before the
  reboot the timeout is 120 seconds. [degr-wfc-timeout]
- If the peer was available before the reboot the timeout will

```

```
expire after 120 seconds. [wfc-timeout]
(These values are for resource 'r0'; 0 sec -> wait forever)
To abort waiting enter 'yes' [ 50]:
```

关闭 SELinux 和 iptables 后, 此错误信息将消失, 两台机器上均可以成功启动 DRBD 服务。
在 centos1 的机器上查看 DRBD 状态, 命令如下:

```
[root@centos1 ~]# service drbd status
```

命令显示结果如下:

```
drbd driver loaded OK; device status:
version: 8.3.13 (api:88/proto:86-96)
GIT-hash: 83ca112086600faacab2f157bc5a9324f7bd7f77 build by mockbuild@builder10.
centos.org, 2012-05-07 11:56:36
m:res cs ro ds p mounted fstype
0:r0 Connected Secondary/Secondary Inconsistent/Inconsistent C
```

之后, 将 centos1 的机器作为 DRBD 的 Primary 机器, 命令如下所示:

```
drbdsetup /dev/drbd0 primary -o
drbdadm primary r0
```

然后再查看其状态, 命令如下所示:

```
[root@centos1 ~]# service drbd status
```

此命令显示结果如下所示:

```
drbd driver loaded OK; device status:
version: 8.3.13 (api:88/proto:86-96)
GIT-hash: 83ca112086600faacab2f157bc5a9324f7bd7f77 build by mockbuild@builder10.
centos.org, 2012-05-07 11:56:36
m:res cs ro ds p mounted fstype
... sync'ed: 0.1% (1429092/1430476)M
0:r0 SyncSource Primary/Secondary UpToDate/Inconsistent C
```

我们发现, Primary/Secondary 关系已形成, 而且数据也正在进行同步, 已同步了 0.1%, 接近 1.5TB 大小的 DRBD 数据同步传输的速度非常慢, 建议将此时间安排在空闲时间, 可将其安排在工作日下班以后, 经过漫长的等待以后, 再查看 Primary 机器的 DRBD 状态, 如下所示:

```
[root@centos1 ~]# service drbd status
```

命令显示结果如下所示:

```
drbd driver loaded OK; device status:
version: 8.3.13 (api:88/proto:86-96)
GIT-hash: 83ca112086600faacab2f157bc5a9324f7bd7f77 build by mockbuild@builder10.
centos.org, 2012-05-07 11:56:36
m:res cs ro ds p mounted fstype
0:r0 Connected Primary/Secondary UpToDate/UpToDate C
```

UpToDate/UpToDate 表示数据已经同步完成了。

关于 DRBD 的性能优化,这里需要说明一下,由于 DELL 机器上的网卡都已经是千兆网卡,建议不要选择百兆交换机,在此例中,由于笔者在两台 DELL 710 机器上已经安装了两条交叉线作为心跳线,所以选择用其中一条心跳线(192.168.1.1->192.168.1.2)作为专门的 DRBD 数据同步线路。

下面将在两台机器上建立 /drbd 分区,以作为 MySQL 的挂载目录,命令如下所示:

```
mkdir /drbd
```

格式化 Primary 机器的 DRBD 分区并挂载使用,命令如下所示:

```
[root@centos1 ~]# mkfs.ext3 /dev/drbd0
```

显示结果如下所示:


```
mke2fs 1.39 (29-May-2006)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
183107584 inodes, 366202530 blocks
18310126 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=4294967296
11176 block groups
32768 blocks per group, 32768 fragments per group
16384 inodes per group
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
    4096000, 7962624, 11239424, 20480000, 23887872, 71663616, 78675968,
    102400000, 214990848

Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information:
done
```

This filesystem will be automatically checked every 28 mounts or 180 days, whichever comes first. Use tune2fs -c or -i to override.

将 /dev/drbd0 设备挂载至 /drbd 分区的命令如下:

```
mount /dev/drbd0 /drbd/
```

 **注意** Secondary 节点上是不允许对 DRBD 设备进行任何操作的,包括只读,所有的读写操作都只能在 Primary 节点上进行,只有当 Primary 节点挂掉时,Secondary 代替主节点作为 Primary 节点后才能进行读写操作。

最后,在两台机器上将 DRBD 设为自启动服务,命令如下:

```
chkconfig drbd on
```

2. Heartbeat 的安装和部署

首先,在两台机器上分别用 yum 来安装 Heartbeat,如下命令操作两次:

```
yum -y install heartbeat
```

如果只操作一次,你会惊奇地发现,第一次时 Heartbeat 并没有安装成功。

然后,在两个节点上配置 Heartbeat 文件。其中,centos1.mypharma.com 的配置文件如下:

```
logfile /var/log/ha-log #定义Heartbeat的日志名字及位置
logfacility local0
keepalive 2             #设定心跳(监测)时间为2秒
deadtime 15             #设定死亡时间为15秒
ucast eth0 112.112.68.172
ucast eth2 10.0.0.2
ucast eth3 192.168.1.2
#采用单播的方式,IP地址指定为对方IP,这里为了防止“脑裂”,特地用了两条心跳线外加公网地址IP作为心跳监测线路,事实上,在项目上线测试阶段,除非人为手动破坏,不然是没有发生“脑裂”的可能。
auto_failback off      #当Primary机器发生故障切换到Secondary机器后不再进行切回操作。
node centos1.mypharma.com centos2.mypharma.com
```

centos2.mypharma.com 的配置文件如下:

```
logfile /var/log/ha-log
logfacility local0
keepalive 2
deadtime 15
ucast eth0 112.112.68.170
ucast eth2 10.0.0.1
ucast eth3 192.168.1.1
auto_failback off
node centos1.mypharma.com centos2.mypharma.com
```

该配置文件跟 centos1 类似,这里不再做重复性解释。

接着,编辑双机互连验证文件 authkeys,命令如下:

```
cat /etc/ha.d/authkeys
auth 1
1 crc
```

需要将此文件的权限设定为 600,不然启动 Heartbeat 服务时会报错,命令如下所示:

```
chmod 600 /etc/ha.d/authkeys
```

最后,编辑集群资源文件 /etc/ha.d/haresources,命令如下:

```
centos1.mypharma.com IPaddr::112.112.68.174/29/eth0 drbdisk::r0 Filesystem::/
dev/drbd0::drbd::ext3 mysqld
```

这个文件在两台机器上是一样的,建议不要轻易改动。

mysqld 为 MySQL 服务器启动、重启及关闭脚本,这个是在安装 MySQL 时自带的,后面在安装 MySQL 时会提到此步。

3. 源码编译安装 MySQL 5.1.47 并部署 haresources

当时之所以选择 MySQL 5.1.47 版本,是因为此版本的 MySQL 在以前的其他项目或网站中均运行稳定,所以在部署此电商项目时也考虑采用此版本,在 MySQL 官方网站上下载 MySQL 5.1.47 的源码包,并在两台机器上分别安装,这里跟单纯编译安装 MySQL 5.1.47 还是略有不同,下面将详细说明一下。

1) 安装 gcc 等基础库文件,命令如下:

```
yum install gcc gcc-c++ zlib-devel libtool ncurses-devel libxml2-devel -y
```

生成运行 MySQL 服务的用户及用户组:

```
groupadd mysql
useradd -g mysql mysql
```

源码编译安装 MySQL 5.1.47:

```
tar zxvf mysql-5.1.47.tar.gz
cd mysql-5.1.47
./configure --prefix=/usr/local/mysql --with-charset=utf8 --with-extra-
charsets=all --enable-thread-safe-client --enable-assembler --with-readline
--with-big-tables --with-plugins=all --with-mysqld-ldflags=-all-static
--with-client-ldflags=-all-static
make
make install
```

2) 对 MySQL 进行权限配置,使其能顺利启动,命令如下:

```
cd /usr/local/mysql
cp /usr/local/mysql/share/mysql/my-medium.cnf /etc/my.cnf
cp /usr/local/mysql/share/mysql/mysql.server /etc/init.d/mysqld
cp /usr/local/mysql/share/mysql/mysql.server /etc/ha.d/resource.d/mysqld
chmod +x /etc/init.d/mysqld
chmod +x /etc/ha.d/resource.d/mysqld
chown -R mysql:mysql /usr/local/mysql
```

3) 在两台机器上的 /etc/my.cnf 的 [mysqld] 项下面重新配置下 MySQL 运行时的数据存放路径。

```
datadir=/drbd/data
```

4) 在 Primary 机器上运行如下命令,使其数据库目录生成数据,Secondary 机器则不需要运行。

```
/usr/local/mysql/bin/mysql_install_db --user=mysql --datadir=/drbd/data
```



注意 这里是整个实验环境的一个重要环节,笔者在搭建时出过几次问题,我们运行 MySQL 是在启动 DRBD 设备之后,即将 /dev/drbd0 目录正确挂载到 /drbd 目录后,而并非尚未挂载就去启动 MySQL,这会导致整个实验完全失败,大家一定要注意。

完成此步骤以后, 不需要启动 MySQL, 它可以靠脚本来启动, 如果已经启动了 MySQL 请手动关闭。

4. 在两台机器上将 DRBD 和 Heartbeat 设成自启动

将 DRBD 和 Heart beat 设成自启动的命令如下:

```
service drbd start
chkcfonig drbd on
service heartbeat start
chkconfig heartbeat on
```

通过观察 Primary 机器上的信息可以得知, Primary 机器已经正确启动了 MySQL 和 Heartbeat, 信息如下所示:

```
[root@centos1 ~]# ip addr
```

此命令显示结果如下所示:

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 84:8f:69:dd:5f:f1 brd ff:ff:ff:ff:ff:ff
    inet 112.112.68.170/29 brd 114.112.69.175 scope global eth0
    inet 112.112.68.174/29 brd 114.112.69.175 scope global secondary eth0:0
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 84:8f:69:dd:5f:f3 brd ff:ff:ff:ff:ff:ff
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 84:8f:69:dd:5f:f5 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.1/24 brd 10.0.0.255 scope global eth2
5: eth3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 84:8f:69:dd:5f:f7 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.1/24 brd 192.168.1.255 scope global eth3
6: virbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
```

通过如下命令查看 3306 端口被占用的情况, 可得知 mysqld 服务已被正常开启。

```
[root@centos1 data]# lsof -i:3306
```

此命令显示结果如下所示:

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE	NODE	NAME
mysqld	4341	mysql	18u	IPv4	9807		TCP	*:mysql (LISTEN)


5. 在 Primary 机器上授权一个远程用户

为 Primary 机器新建一个授权远程用户, 例如这里的 admin, 方便程序进行连接, 命令如下所示:

```
mysql> grant all privileges on *.* to 'admin'@'%' identified by 'admin@change20110101';
```

修改 PHP 程序连接 MySQL 的配置文件，如 config.inc.php 或 configuration.php，修改代码如下：

```
public $host = '112.112.68.174';
public $user = 'admin';
public $password = 'admin@change20120101';
```

 **注意** 此时 PHP 程序连接的应该的是 Heartbeat 产生的 VIP 地址，即 112.112.68.174。

6. 测试工作

这里主要是模拟 Primary 机器重启或宕机时，测试 Secondary 机器能不能自动接管过来并启动 MySQL，我们重启 Primary 机器后又再重启 Secondary 机器，正常的结果应该是：无论我们如何重启机器，只要保证有一台机器存活，MySQL 均能提供正常的服务。在 Primary 机器上可以通过 tail 命令进行观察，命令如下所示：

```
[root@centos1 ~]# tail -n 100 /var/log/messages
```

结果如下所示：

```
Oct 12 15:19:44 centos1 heartbeat: [4074]: info: Link centos2.mypharma.com:eth2 up.
Oct 12 15:19:44 centos1 heartbeat: [4074]: info: Status update for node centos2.
mypharma.com: status init
Oct 12 15:19:44 centos1 heartbeat: [4074]: info: Status update for node centos2.
mypharma.com: status init
Oct 12 15:19:44 centos1 heartbeat: [4074]: info: Link centos2.mypharma.com:eth3
up.
Oct 12 15:19:44 centos1 heartbeat: [4074]: info: Status update for node centos2.
mypharma.com: status up
Oct 12 15:19:44 centos1 harc[6268]: info: Running /etc/ha.d/rc.d/status status
Oct 12 15:19:44 centos1 harc[6284]: info: Running /etc/ha.d/rc.d/status status
Oct 12 15:19:44 centos1 heartbeat: [4074]: info: Status update for node centos2.
mypharma.com: status active
Oct 12 15:19:44 centos1 harc[6300]: info: Running /etc/ha.d/rc.d/status status
Oct 12 15:19:45 centos1 heartbeat: [4074]: info: remote resource transition
completed.
Oct 12 15:19:46 centos1 heartbeat: [4074]: info: Link centos2.mypharma.com:eth0
up.
```

待系统稳定以后，隔段时间用命令抽查一下 Heartbeat 日志，命令如下：

```
[root@centos1 ~]# tail -n 100 /var/log/ha-log
```

此命令显示结果如下所示：

```
heartbeat[4063]: 2012/10/15_16:57:46 info: cl_malloc stats:668/1965676344080/21163
```



```

44080/21163[pid4085/HBWRITE]
heartbeat[4063]: 2012/10/16_16:57:59 info: RealMalloc stats: 52392 total malloc
bytes. pid [4085/HBWRITE]
heartbeat[4063]: 2012/10/16_16:57:59 info: Current arena value: 0
heartbeat[4063]: 2012/10/16_16:57:59 info: MSG stats: 0/0 ms age 4639692810
[pid4086/HBREAD]
heartbeat[4063]: 2012/10/16_16:57:59 info: cl_malloc stats: 360/414819
52960/25724 [pid4086/HBREAD]
heartbeat[4063]: 2012/10/16_16:57:59 info: RealMalloc stats: 45564 total malloc
bytes. pid [4086/HBREAD]
heartbeat[4063]: 2012/10/16_16:57:59 info: Current arena value: 0
heartbeat[4063]: 2012/10/16_16:57:59 info: These are nothing to worry about.
heartbeat[4063]: 2012/10/17_16:58:12 info: Daily informational memory statistics
heartbeat[4063]: 2012/10/17_16:58:12 info: MSG stats: 6/1036311 ms age 0 [pid4063/
MST_CONTROL]
heartbeat[4063]: 2012/10/17_16:58:12 info: cl_malloc stats: 578/32771436
115384/56399 [pid4063/MST_CONTROL]
heartbeat[4063]: 2012/10/17_16:58:12 info: RealMalloc stats: 597304 total malloc
bytes. pid [4063/MST_CONTROL]
heartbeat[4063]: 2012/10/17_16:58:12 info: Current arena value: 0
heartbeat[4063]: 2012/10/17_16:58:12 info: MSG stats: 0/4 ms age 430777840
[pid4080/HBFIFO]
heartbeat[4063]: 2012/10/17_16:58:12 info: cl_malloc stats: 312/413 36320/16115
[pid4080/HBFIFO]
heartbeat[4063]: 2012/10/17_16:58:12 info: RealMalloc stats: 38892 total malloc
bytes. pid [4080/HBFIFO]
heartbeat[4063]: 2012/10/17_16:58:12 info: Current arena value: 0
heartbeat[4063]: 2012/10/17_16:58:12 info: MSG stats: 0/0 ms age 4726092800
[pid4081/HBWRITE]
heartbeat[4063]: 2012/10/17_16:58:12 info: cl_malloc stats: 332/273119
39824/18379 [pid4081/HBWRITE]
heartbeat[4063]: 2012/10/17_16:58:12 info: RealMalloc stats: 48136 total malloc
bytes. pid [4081/HBWRITE]
heartbeat[4063]: 2012/10/17_16:58:12 info: Current arena value: 0
heartbeat[4063]: 2012/10/17_16:58:12 info: MSG stats: 0/0 ms age 4726092800
[pid4082/HBREAD]
heartbeat[4063]: 2012/10/17_16:58:12 info: cl_malloc stats: 333/518422
39916/18443 [pid4082/HBREAD]
heartbeat[4063]: 2012/10/17_16:58:12 info: RealMalloc stats: 40740 total malloc
bytes. pid [4082/HBREAD]
heartbeat[4063]: 2012/10/17_16:58:12 info: Current arena value: 0
heartbeat[4063]: 2012/10/17_16:58:12 info: MSG stats: 0/0 ms age 4726092800
[pid4083/HBWRITE]
heartbeat[4063]: 2012/10/17_16:58:12 info: cl_malloc stats: 344/273149
41952/19771 [pid4083/HBWRITE]
heartbeat[4063]: 2012/10/17_16:58:12 info: RealMalloc stats: 50264 total malloc
bytes. pid [4083/HBWRITE]
heartbeat[4063]: 2012/10/17_16:58:12 info: Current arena value: 0
heartbeat[4063]: 2012/10/17_16:58:12 info: MSG stats: 0/0 ms age 4726092800
[pid4084/HBREAD]

```

```

heartbeat[4063]: 2012/10/17_16:58:12 info: cl_malloc stats: 356/518505
44400/21427 [pid4084/HBREAD]
heartbeat[4063]: 2012/10/17_16:58:12 info: RealMalloc stats: 45748 total malloc
bytes. pid [4084/HBREAD]
heartbeat[4063]: 2012/10/17_16:58:12 info: Current arena value: 0
heartbeat[4063]: 2012/10/17_16:58:12 info: MSG stats: 0/0 ms age 4726092800
[pid4085/HBWRITE]
heartbeat[4063]: 2012/10/17_16:58:12 info: cl_malloc stats: 356/273179
44080/21163 [pid4085/HBWRITE]
heartbeat[4063]: 2012/10/17_16:58:12 info: RealMalloc stats: 52392 total malloc
bytes. pid [4085/HBWRITE]
heartbeat[4063]: 2012/10/17_16:58:12 info: Current arena value: 0
heartbeat[4063]: 2012/10/17_16:58:12 info: MSG stats: 0/0 ms age 4726092800
[pid4086/HBREAD]
heartbeat[4063]: 2012/10/17_16:58:12 info: cl_malloc stats: 360/518499
52960/25724 [pid4086/HBREAD]
heartbeat[4063]: 2012/10/17_16:58:12 info: RealMalloc stats: 45564 total malloc
bytes. pid [4086/HBREAD]
heartbeat[4063]: 2012/10/17_16:58:12 info: Current arena value: 0
heartbeat[4063]: 2012/10/17_16:58:12 info: These are nothing to worry about.

```

最后一句话表示整个 Heartbeat+DRBD+MySQL 环境是稳定的，没什么可以担心的了，笔者负责的网站已经稳定运行了 1059 天，DRBD+Heartbeat 也没有任何异常，用 uptime 命令观察可以得知，系统正常稳定，命令如下所示：

```
uptime
```

命令显示如果如下：

```
15:21:30 up 159 days, 22:25, 1 user, load average: 0.11, 0.04, 0.01
```

不过，DRBD+Heartbeat 也存在着一定的问题，下面来看看。

- “脑裂”问题，这个是大家在工作中讨论得最多的问题，但是在网站的维护工作中我们发现，由于 NFS 服务器和 MySQL 采用的都是 DRBD 双机，因此只要不轻易触碰机器的交叉线（即心跳线），保证交换机的稳定性，出现此问题的概率几乎微乎其微。
- I/O 消耗，这个是由 DRBD 本身的写文件机制造成的，我们通过 Sysbench 做基准测试时也能发现此问题，建议将除系统表之外的所有表引擎都换成 InnoDB 引擎，避免 DRBD 写机制对数据库中的表的影响。如果觉得这样太麻烦，可以直接采用 MySQL5.5 或更高级别的 MySQL 版本（因为它们的默认引擎就是 InnoDB）。
- DRBD+Heartbeat 浪费了宝贵的服务器资源，DRBD 的备机目前还不能提供读功能，生产环境下的 MySQL 服务器硬件基本上都是顶配的，这样就浪费了一台机器资源，感觉还是非常可惜的。

下面附上 MySQL 中 MyISAM 转 InnoDB 的 Shell 脚本，脚本如下：

```
#!/bin/bash
```

```
#Date:2012/09/27
DB=pharma
USER=root
PASSWD=root@change

/usr/local/mysql/bin/mysql -u$USER -p$PASSWD $DB -e "select TABLE_NAME from
information_schema.TABLES where TABLE_SCHEMA='$DB' and ENGINE='MyISAM';"
| grep -v "TABLE_NAME" > mysql_table.txt
#for t_name in `cat tables.txt`
do
cat mysql_table.txt | while read LINE
do
echo "Starting convert table engine..."
/usr/local/mysql/bin/mysql -u$USER -p$PASSWD $DB -e "alter table $LINE
engine='InnoDB'"
sleep 1
done
```

整个实施过程需要注意以下几点:

- ❑ 在 Secondary 主机上, 用来做 DRBD 的硬盘可以跟 Primary 主机的大小不一样, 但请不要小于 Primary 主机, 以免发生数据丢失的现象, 生产环境下建议保持大小一致, 如果确实不能保持一样的大小, Secondary 机器的 DRBD 分区应大于 Primary 机器。
- ❑ 服务器网卡及交换机推荐采用千兆系列的, 在测试中发现其同步速率带宽介于 100MB~200MB, 这里采用官方的建议, 以带宽最小值的 30% 来设置带宽, 即 100MB*30%, 大家也可根据自己的实际网络环境来设定此值。
- ❑ DRBD 对网络的环境的要求很高, 建议用单独的交叉线来作为两台主机之间的心跳线, 如果条件允许, 可以考虑用两根以上的心跳线; 如果这个环节做得好, 基本上“脑裂”的问题是不存在的。其实整个实验初期都可以在同一网络环境下实现, 后期再加心跳线也是可行的。
- ❑ 安装 Heartbeat 时需要安装两遍, 即 `yum -y install heartbeat` 要执行两次。
- ❑ 建议不要用根分区作为 MySQL 的 `datadir`, 不然执行 `show database` 命令时会发现了名为“`#mysql50#lost+found`”的数据库, 这也是笔者将 MySQL 的数据库目录设置成 `/drbd/data` 的原因。
- ❑ 就算发生“脑裂”的问题, DRBD 也不会丢失数据, 手动解决此脑裂问题即可, 况且用两根或两根以上的心跳线的话, 出现“脑裂”的概率非常之少; 正因为 DRBD 可靠, MySQL 也推荐将其作为 MySQL 实现高可用方案之一。
- ❑ MySQL 的 DRBD 此方案不能达到毫秒级的切换速度, MyISAM 引擎的表在系统宕机后需要很长的修复时间, 而且也有可能发生表损坏的情况, 建议大家将除了系统表之外的所有表引擎均改为 InnoDB 引擎。

7.4.6 案例分享六：生产环境下的 MySQL 数据库主从同步

MySQL 的主从 Replication 同步（又叫主从复制）是一个很成熟的架构（如图 7-9 所示），笔者负责的许多电商平台的线上环境采用的都是这种方案，它的优点为：

- 在业务繁忙的阶段，可以在从服务器上执行查询工作（即我们常说的读写分离），降低主服务器的压力。
- 在从服务器上进行备份，避免备份期间影响主服务器服务。
- 当主服务器出现问题时，可以迅速切换到从服务器，这样就不会影响线上环境了。
- 数据分布。由于 MySQL 复制并不需要很大的带宽，所以可以在不同的数据中心实现跨机房数据的复制（阿里巴巴这种级别的公司业务则另当别论，大家可以关注他们的 otter 分布式数据库系统）。比如笔者目前的数据中心的 Adserver 业务数据库，就是一主四从。

主从复制是 MySQL 数据库提供的一种高可用、高性能的解决方案，其实现并不复杂，它不是完全实时的，而是异步实时的，如果网络延迟比较严重，则要考虑将其延迟时间作为 Nagios 报警的选项参数，其具体工作步骤为：

- 1) 主服务器把数据更新记录到二进制日志中。
- 2) 从服务器把主服务器的二进制日志复制到自己的中继日志中，这个由从服务器的 I/O 线程负责。
- 3) 从服务器执行中继日志，将其更新应用到自己的数据库上，这个由从服务器的 SQL 线程负责。

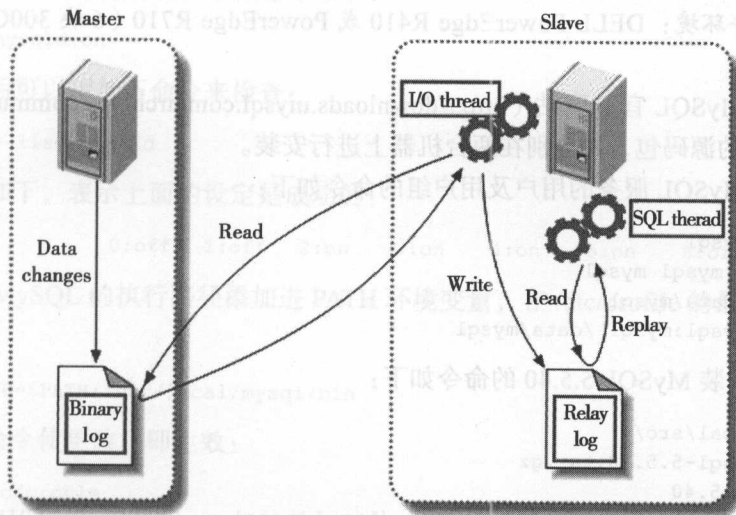


图 7-9 MySQL 数据库主从同步的工作流程

鉴于生产环境下对 MySQL 有更严谨的要求，推荐采用源码编译的方法安装 MySQL 数据库。以下内容笔者曾在个人博客中阐述过，并且收到了许多热心网友的中肯意见，笔者

据此进行了 5 次修改。如果大家在工作中有相应的需求，可以参考下。

编译安装 MySQL 数据库之前建议在服务器上安装基础库文件，命令如下：

```
yum -y install gcc gcc-c++ autoconf libjpeg libjpeg-devel libpng libpng-devel
freetype freetype-devel libxml2 libxml2-devel zlib zlib-devel glibc glibc-devel
glib2 glib2-devel bzip2 bzip2-devel ncurses ncurses-devel curl curl-devel e2fsprogs
e2fsprogs-devel krb5 krb5-devel libidn libidn-devel openssl openssl-devel cmake
```

由于服务器采用的是最小化安装，因此建议也安装一下开发工具和开发库，以防止源码编译安装 MySQL 时报错，另外，MySQL 从 5.5 版本开始，通过 `./configure` 进行编译配置的方式已经被取消，取而代之的是 `cmake` 工具，因此，我们首先要在系统中源码编译安装 `cmake` 工具。

MySQL 数据库涉及的文件及对应的目录如下。

MySQL 的安装位置：`/usr/local/mysql`

MySQL 的配置文件：`/etc/my.cnf`

MySQL 数据库的位置：`/data/mysql/`

下面介绍一下工作环境。

主数据库 IP：192.168.1.205

从数据库 IP：192.168.1.204

系统：CentOS 6.4 x86-64

内核版本：2.6.32-358.el6.x86_64

MySQL 版本：5.5.40

服务器硬件环境：DELL PowerEdge R410 或 PowerEdge R710（6 块 300G SAS300G 做成 RAID 10）

首先，在 MySQL 官方网站（<http://downloads.mysql.com/archives/community/>）上下载 MySQL 5.5.40 的源码包，并分别在两台机器上进行安装。

生成运行 MySQL 服务的用户及用户组的命令如下：

```
groupadd mysql
useradd -g mysql mysql
mkdir -p /data/mysql
chown -R mysql:mysql /data/mysql
```

源码编译安装 MySQL 5.5.40 的命令如下：

```
cd /usr/local/src/
tar xvf mysql-5.5.40.tar.gz
cd mysql-5.5.40
cmake -DCMAKE_INSTALL_PREFIX=/usr/local/mysql -DMYSQL_DATADIR=/data/mysql
-DDEFAULT_CHARSET=utf8 -DDEFAULT_COLLATION=utf8_unicode_ci -DWITH_READLINE=1
-DWITH_SSL=system -DWITH_EMBEDDED_SERVER=1 -DENABLED_LOCAL_INFILE=1
-DDEFAULT_COLLATION=utf8_general_ci -DWITH_MYSQL_STORAGE_ENGINE=1 -DWITH_
INNOBASE_STORAGE_ENGINE=1 -DWITH_DEBUG=0
make && make install && cd ../
```


然后,对 MySQL 进行权限配置,将 MySQL 的数据安装路径设为 /data/mysql,并配置 MySQL 为服务启动状态,命令如下:

```
cd /usr/local/src/mysql-5.5.40
cp ./support-files/my-huge.cnf /etc/my.cnf
cp ./support-files/mysql.server /etc/init.d/mysqld
chmod +x /etc/init.d/mysqld
chown -R mysql:mysql /usr/local/mysql
```

这里暂时只用系统自带的“my-huge.cnf”作为 MySQL 数据库的配置文件,后期将根据 MySQL 的 status 运行状态来进行调优整理。

第三步是修改这两台服务器的 MySQL 服务器下的 /etc/my.cnf 的 [mysqld] 项,在其下面添加 MySQL 运行时的数据存放路径,命令如下所示:

```
datadir=/data/mysql
```

第四步是在这两台服务器上分别运行如下命令初始化数据库,在 /data/mysql 下生成 MySQL 的初始化文件和初始库等。

```
/usr/local/mysql/scripts/mysql_install_db --user=mysql --basedir=/usr/local/
mysql --datadir=/data/mysql
```

此时就可以顺利地以服务的形式来启动 MySQL 服务了,命令如下:

```
service mysqld start
```

现在将其配置成为自启动状态,命令如下:

```
chkconfig mysqld on
```

配置完成后可以用如下命令来检查:

```
chkconfig --list mysqld
```

结果显示如下,表示上面的设定是成功的:

```
mysqld          0:off  1:off  2:on   3:on   4:on   5:on   6:off
```

最后,将 MySQL 的执行路径添加进 PATH 环境变量,在 /etc/profile 的最后一行添加如下内容:

```
export PATH=$PATH:/usr/local/mysql/bin
```

执行如下命令使更改立即生效:

```
source /etc/profile
```

下面详细介绍一下主从复制同步的过程。

1. 设置主库

1) 修改主库 my.cnf,主要是设置各不一样的 server-id,以及要同步的数据库名字。可以用 vim 编辑 /etc/my.cnf 文件,在 [mysqld] 段下面增加如下内容:

```
server-id = 1
log-bin= binlog
binlog_format=mixed
```

从库 my.cnf 文件跟主库不一样，具体改动如下：

```
server-id=2
log-bin= binlog
binlog_format=mixed
replicate_wild_do_table=adserver.%
replicate_wild_ignore_table=mysql.%
```

adserver 为要同步的数据库的名字，mysql 为不需要同步的数据库，这里为了避免发生跨库同步失败的问题，建议在从库里面这样配置。将从库所在服务器的 server-id 修改为非 1 的数字即可，不然会在同步的过程中发现错误，主库所在的机器的是必须要开启二进制日志的，其日志格式为 mixed，而从库服务器则不必非要开启二进制日志。

2) 分别重启主从库服务器让修改的配置生效，命令如下：

```
service mysqld restart
```

3) 登录主库：

```
mysql -u root -p
```

此处不必输入密码即可进行，这是由于 MySQL 服务器是刚配置的，所以 MySQL 数据库的 root 密码暂时没有配置，为了安全起见，大家可以稍后再进行设置。

4) 赋予从库权限账号，允许用户在主库上读取日志，命令如下：

```
mysql> grant replication slave on *.* to 'admin'@'192.168.11.27' identified by
'admin@101';
```

replication slave 是一个基本的必须的权限，它直接授予从机服务器以该账户连接主机执行 replication 操作的权限，为了安全起见，建议只分配 admin 账户 replication slave 权限。

此操作完成以后，建议立即在从机上验证一下，如果成功显示 MySQL 登录界面则表示设置成功，命令如下：

```
mysql -u admin -p -h 192.168.1.204
Enter password:
```

输入 admin 密码以后应该有如下成功登录的显示结果。

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 5.5.40-log Source distribution
Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
```

这里有个知识点需要说明一下，MySQL 数据库的权限系统在实现上比较简单，相关权限信息主要存储在几个被称为 grant tables 的系统表中，即：mysql.user、mysql.db、mysql.host、mysql.table_priv 和 mysql.column_priv。由于权限信息的数据量比较小，访问又非常频繁，所以 MySQL 在启动的时候，就会将所有的权限信息都加载到内存中，并保存在几个特定的结构里。这就使得每次在手动修改了相关权限表之后，都必须执行 flush privileges，通知 MySQL 重新加载 MySQL 的权限信息。当然，如果通过 grant、revoke 或 drop user 命令来修改相关权限，则不必再手动执行 flush privileges 命令了。

5) 检查创建是否成功，命令显示结果如下（我们只需关注 user 名为 admin 的那行即可）：

```
mysql> select user,host from mysql.user;
```

```
+-----+-----+
| user | host |
+-----+-----+
| root | 127.0.0.1 |
| admin | 192.168.1.205 |
| root | ::1 |
|      | fabric |
| root | fabric |
|      | localhost |
| root | localhost |
+-----+-----+
7 rows in set (0.01 sec)
```

6) 锁主库表，命令如下（建议不要退出此终端，以免锁表失败）：

```
mysql> flush tables with read lock;
```

7) 显示主库信息。

这里要记录一下 File 和 Position，因为在设置从库时将会用到它们，命令如下：

```
mysql> show master status;
```

```
+-----+-----+-----+-----+
| File | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.000004 | 106 | mydata | |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

8) Slave 端机器获取 Master 端 MySQL 数据库的“快照”有两种方法。第一种是锁表后直接用 tar 将原 Master 打包给 Slave 机器，这种情况适用于网站初始化，而且数据库比较单一的工作场景。比如说笔者的数据库已经在线运行了一段时间，由于 MySQL 中有默认设置，因此没有按表空间分离数据，所有的表数据都被放到 ibdata1 文件中了，其中包括数据字典（也就是 InnoDB 表的元数据）、变更缓冲区、双写缓冲区、撤销日志。ibdata1 文件的大小会持续增长，就算删除了表的数据，ibdata1 文件的体积也不会减小。所以如果是采用

tar 方案的话，首先需要对 ibdata1 文件进行“瘦身”操作。

如果机器上还存在着不同的数据库，那么就不适合用 tar 打包的方法了，比如笔者的订单系统的数据库上面的生产数据库数据差不多有 9.8GB 了，321 张表，但有的数据库不需要同步，所以可以用别的方法来进行（比如 mysqldump，它可以单独对某个特定的数据库进行逻辑备份，下面会重点讲解这种方法）。

如果只需单独备份主机上的 adserver 数据库，可以用如下的方式：

```
mysqldump --master-data -u root -p adserver > adserver.sql
```

另外，稍微解释一下 --master-data 参数的作用，mysqldump 程序的开发者给程序设计这项参数是为了帮助获取对应的 Log Position，在添加了这个参数选项以后，mysqldump 会在 dump 文件中产生一条 CHANGE MASTER TO 命令，命令中记录了 dump 时刻所对应的 Log Position 的详细信息。

2. 设置从库

1) 在主库上将 adserver.sql 传输给从机，推荐用 rsync 命令，如果是生产环境下的数据库，那么 rsync 尤其适合在服务器之间传输上百 GB 的生产数据库数据，命令如下：

```
rsync -vzrtopg adserver.sql root@192.168.1.205:/root/
```

2) 登录从库，建立 adserver 数据库，命令如下：

```
mysql> create database adserver;
```

然后，退出 mysql 命令行，导入 adserver.sql 数据，命令如下：

```
mysql -u root -p adserver < /root/adserver.sql
```

3) 解锁主库表，命令如下所示：

```
mysql> unlock tables;
```

4) 在从库上设置同步。

设置连接 MASTER “MASTER_LOG_FILE” 为主库的 File，“MASTER_LOG_POS” 为主库的 Position，命令如下所示：

```
mysql> slave stop;
mysql> change master to master_host='192.168.1.204',master_user='admin', master_
password='admin@101',
master_log_file='mysql-bin.000004', master_log_pos=7323251;
mysql> slave start;
```

5) 查看从库的 status 状态，命令如下所示：

```
mysql> show slave status\G;
```

结果如下所示：

```
***** 1. row *****
```

```

Slave_IO_State: Waiting for master to send event
Master_Host: 192.168.1.204
Master_User: admin
Master_Port: 3306
Connect_Retry: 60
Master_Log_File: mysql-bin.000004
Read_Master_Log_Pos: 7323251
Relay_Log_File: localhost-relay-bin.000002
Relay_Log_Pos: 253
Relay_Master_Log_File: mysql-bin.000004
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Do_DB:
Replicate_Ignore_DB:
Replicate_Do_Table:
Replicate_Ignore_Table:
Replicate_Wild_Do_Table: adserver.%
Replicate_Wild_Ignore_Table: mysql.%
Last_Errno: 0
Last_Error:
Skip_Counter: 0
Exec_Master_Log_Pos: 7323251
Relay_Log_Space: 413
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
Last_IO_Errno: 0
Last_IO_Error:
Last_SQL_Errno: 0
Last_SQL_Error:
Replicate_Ignore_Server_Ids:
Master_Server_Id: 1
1 row in set (0.00 sec)

ERROR:
No query specified

```

注意上面显示结果中两部分的显示内容，“Slave_IO_Running: Yes”表示网络正常，“Slave_SQL_Running: Yes”表示结构正常，根据 MySQL 主从同步的原理，这两个部分必须都为 YES（正常）才表示同步是成功的，此外，还要注意 Seconds_Behind_Master 这个选项，它表示主从同步延迟时间，在一些对数据即时性要求很高的生产场景下，这个选项也



应该引起我们足够的重视。

6) 进行一些测试工作。

在主库的 adserver 数据上建立名为 yuhongchun 的表, 命令如下:

```
mysql> CREATE TABLE `yuhongchun` (
  `id` INT(5) UNSIGNED NOT NULL AUTO_INCREMENT,
  `username` VARCHAR(20) NOT NULL,
  `password` CHAR(32) NOT NULL,
  `time` DATETIME NOT NULL,
  `number` FLOAT(10) NOT NULL,
  `content` TEXT NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE = MYISAM;
```

在从机中马上就可以看到, 在 mydata 数据库下产生了名为 yuhongchun 的表, 只不过表的记录目前为空, 另外, 观察下从机的中继日志, 可以用如下命令:

```
mysqlbinlog localhost-relay-bin.000002
```

结果显示如下所示:

```
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=1*/;
/*!40019 SET @@session.max_insert_delayed_threads=0*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
DELIMITER /*!*/;
# at 4
#150707 4:16:16 server id 2  end_log_pos 107  Start: binlog v 4, server v
5.5.40-log created 150707 4:16:16
BINLOG '
0IqbVQ8CAAAAZwAAAAgSAAAAAAQANS41LjQwLWxvZwAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAEzgNAAgAEgAEBAQEgAAVAAGggAAAAICAgCAA==
'/*!*/;
# at 107
#691231 19:00:00 server id 1  end_log_pos 0  Rotate to mysql-bin.000004 pos:
7323251
# at 150
#150707 3:11:08 server id 1  end_log_pos 0  Start: binlog v 4, server v
5.5.40-log created 150707 3:11:08
BINLOG '
jHubVQ8BAAAAZwAAAAAAAAAAAAQANS41LjQwLWxvZwAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAEzgNAAgAEgAEBAQEgAAVAAGggAAAAICAgCAA==
'/*!*/;
# at 253
#150707 4:20:17 server id 1  end_log_pos 7323578  Query thread_id=12
exec_time=0 error_code=0
use `adserver`/*!*/;
SET TIMESTAMP=1436257217/*!*/;
SET @@session.pseudo_thread_id=12/*!*/;
SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=0, @@session.
unique_checks=1, @@session.autocommit=1/*!*/;
```

```

SET @@session.sql_mode=0/*!*/;
SET @@session.auto_increment_increment=1, @@session.auto_increment_offset=1/*!*/;
/*!C utf8 *//*!*/;
SET @@session.character_set_client=33,@@session.collation_connection=33,@@session.collation_server=33/*!*/;
SET @@session.lc_time_names=0/*!*/;
SET @@session.collation_database=DEFAULT/*!*/;
CREATE TABLE `yuhongchun` (
  `id` INT(5) UNSIGNED NOT NULL AUTO_INCREMENT,
  `username` VARCHAR(20) NOT NULL,
  `password` CHAR(32) NOT NULL,
  `time` DATETIME NOT NULL,
  `number` FLOAT(10) NOT NULL,
  `content` TEXT NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE = MYISAM
/*!*/;
DELIMITER ;
# End of log file
ROLLBACK /* added by mysqlbinlog */;
/*!50003 SET COMPLETION_TYPE=@OLD_COMPLETION_TYPE*/;
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=0*/;

```

这表明主从同步复制是成功的。

3. MySQL 主从同步常见的问题

熟悉 MySQL 主从复制架构的朋友应该很熟悉常见的错误，其实主要就是网络、MySQL 访问权限、iptables 和 SELinux 等问题，我们平时注意检查这些问题，处理起来应该不是很困难，由于我们一般将 MySQL 主从服务器放在局域网（内网）环境中，在局域网环境中我们要记得关闭 iptables 和 SELinux，注意 Slave_IO_Running 和 Slave_SQL_Running 的状态必须确保为 YES 才行，另外也要注意从机的 Seconds_Behind_Master 值。

这里跟大家分享下笔者在工作中主从同步时遇到的问题。

(1) 表结构不同导致的主从同步失败的问题，从机报错信息如下所示：

```

120307 14:44:50 [ERROR] Slave: Error 'Cannot add or update a child row: a
foreign key constraint fails ('offer99/fulfillment', CONSTRAINT `fulfillment_ibfk_1` FOREIGN KEY (`offer_id`) REFERENCES `offer` (`offer_id`))' on
query. Default database: 'offer99'. Query: 'INSERT INTO fulfillment(account_id,product_id,f_record_ts,fulfilled_ts,transaction_id,statuscode,gross_rev_in,gross_rev_out,merch_rev,net_rev,vc_points,vc_exchange_rate,offer_id,tab_name,advertising) VALUES('9496896','613','2012-03-07 09:33:46','2012-03-09:30:02','10139126','1','20.00','12.00','12.00','8','120000','10000.00','663','-1','1')', Error_code: 1452

```

笔者起初以为是外键约束的问题，在这上面浪费了不少时间；在这里跟大家提个醒，

这个时候千万不要盲目地删除外键，这会导致在以后的 MySQL 主从同步维护工作中后患无穷，是个治标不治本的方法。这个时候更应该静下心来处理这个问题，我仔细地检查了两边数据库的 fulfillment 和 offer 表，发现两边的 offer 表数据不一致，特别是从数据库，差了 3 条数据，那么我们如何从几万条数据中找到究竟是差了哪 3 条数据呢？

这里可以用 select into outfile 的方法导出两边 offer 表的数据，将其保存为 CSV 格式的文件，然后用 Linux 下的 diff 命令进行比较，这样很快就能找出差了的数据的 offer_id 值，然后以 SQL 格式的形式导出数据并导进从机（load 命令也是可以的），然后在从机上重新执行 slave start 命令进行同步，这样就可以排除故障了。

```
SELECT * FROM offer WHERE offer_id IN (658,663,694);
```

（2）表结构不同导致的从机更新失败

根据 MySQL 官方文档的说明，如果遇到错误的 SQL 执行语句的时候，故障的表象是从机不会去同步主库，所以要手工地让这个语句不被执行，跳过 N 个事件步骤后直接处理下一个事件，而这个跳过去的事件对数据的完整性是没有什么影响的。一般设置“SET GLOBAL sql_slave_skip_counter = 1”就可以跳过去了，如果跳不过去，就要具体判断得跳多少步才能正确了。

这里再提供一个具体案例以供大家参考，我们本来就在从机上安装了 bugfree 数据库，结果笔者不小心在主数据库上也安装了 bugfree 数据库，这时候从机上不能再创建 bugfree 数据库了，直接导致 Replication 同步失败。这个时候可以用“set global sql_slave_skip_counter = N”忽略这个操作（N 值取决于在主机上建立 bugfree 库和表的值），从而让数据库主从同步正常。

（3）主机硬件故障，如何切换主从服务器（一主多从）

某天早上笔者一来就发现公司的网站打不开了，初步断定是主数据库出了问题，发现 SSH 都登录不上去了，这时候紧急联系机房人员让他们帮忙重启，发现居然连重启都重启不了，他们断定是 MySQL 服务器硬件出了问题。这时候需要紧急将从机提升为主服务器，我们应该如何操作呢？可以按照如下的步骤进行操作：

- 1) 用 stop slave IO_THREAD 命令在从机上停掉 IO_Thread 进程，确保从机上再没有同步的 SQL 语句，即出现“Has read all relay log”语句字样。

- 2) 在从数据库上执行 stop slave 命令停止从机服务，然后执行 reset master 命令将其设置成主数据库。

- 3) 在起始的从机上将原有的主机 IP 地址更换为此机器（现在已变为 Master 主数据库）的 IP 地址。

- 4) 删除新的主数据库服务器的 master.info 和 relay-log.info 文件，防止它下次重启时还是按照从机来启动。

MySQL 主从 Replication 复制非常快，在保证网络的前提下，小数据的改变几乎感觉

不到延迟（但还是属于异步同步），通常在 Master 端改动以后，Slave 端也会立即改动，这种模式非常适合那种对延时性要求很低的工作环境，比如线上的 BBS 论坛；如果是电子商务网站，由于对数据的即时性要求很高，建议不要用读写分离的方案，即：所有的读写操作均在主机上实现，从机只是作为主机的备份。其他非电子商务性质的网站，可以考虑 MySQL 一主多从，读写分离的方案。

7.4.7 案例分享七：HAProxy 双机高可用方案之 HAProxy+Keepalived

由于笔者公司的注册用户已经超过 500 万了，而且每天都有持续增长的趋势，因此 PV/日已经有向千万/日靠近的趋势了，原有的 Web 架构越来越满足不了公司的需求了，所以我们也考虑用能抗高并发的 HAProxy 来作为网站最前端的负载均衡器。因为之前笔者已经在东莞的两个金融项目上面成功地实施了 HAProxy+Keepalived 双机方案，所以这里也想尝试在公司的 CPA 电子广告平台上采用这种负载均衡高可用架构，即 HAProxy+Keepalived。

1. 做好整个环境的准备工作

两台服务器 DELL 2950 均要提前做好准备工作，比如设置好 hosts 文件及进行 NTP 对时。

网络拓扑很简单，如下所示：

```
ha1.offer99.com eth0:203.93.236.145
ha2.offer99.com eth0:203.93.236.142
```

网卡用其自带的千兆网卡即可。

硬盘模式没有要求，RAID0 或 RAID1 均可，单硬盘其实也是可以的。

网站对外的 VIP 地址是：203.93.236.149，这是通过 Keepalived 来实现的，原理请参考前面的章节；同时这也是网站的外网 DNS 对应的 IP。

在这里，HAProxy 采用 7 层模式，Frontend（前台）根据任意 HTTP 请求头内容进行规则匹配，然后把请求定向到相关的 Bankend（后台）。

2. HAProxy 和 Keepalived 的安装过程

安装过程前面已经介绍过，这里不再复述，网站上线的时候用的是当前 HAProxy 的最新版本 1.4.18。

首先在两台负载均衡机器上启动 HAProxy，命令如下：

```
/usr/local/haproxy/sbin/haproxy -c -q -f/usr/local/haproxy/conf/haproxy.cfg
```

如果启动了 HAProxy 程序后，我们又修改了 haproxy.cfg 文件，则可以用如下命令平滑重启 HAProxy 让新配置文件生效，命令如下所示：

```
/usr/local/haproxy/sbin/haproxy -f /usr/local/haproxy/conf/haproxy.conf -st
`cat /usr/local/haproxy/haproxy.pid`
```


新版的 HAProxy 支持 reload 命令，即大家可以用“service haproxy reload”命令来重载 HAProxy。

这里将以网站的生产环境下的配置文件 /usr/local/haproxy/conf/haproxy.cfg 为例进行说明，其具体内容如下所示（两台 HAProxy 机器的配置内容是一样的）：

```
global
    maxconn 65535
    chroot /usr/local/haproxy
    uid 99
    gid 99
    #maxconn 4096
    spread-checks 3
    daemon
    nbproc 1
    pidfile /usr/local/haproxy/haproxy.pid

defaults
    log 127.0.0.1 local3
    mode http
    option httplog
    option httpclose
    option dontlognull
    option forwardfor
    option redispatch
    retries 10
    maxconn 2000
    stats uri /haproxy-stats
    stats auth admin:admin
    timeout 5000
    clitimeout 50000
    srvtimeout 50000

frontend HAProxy
    bind *:80
    mode http
    option httplog
    acl cache_domain path_end .css .js .gif .png .swf .jpg .jpeg
    acl cache_dir path_reg /appimg
    acl cache_jpg path_reg /theme
    acl bugfree_domain path_reg /bugfree

    use_backend varnish.offer99.com if cache_domain
    use_backend varnish.offer99.com if cache_dir
    use_backend varnish.offer99.com if cache_jpg
    use_backend bugfree.offer99.com if bugfree_domain
    default_backend www.offer99.com

backend bugfree.offer99.com
    server bugfree 222.35.135.151:80 weight 5 check inter 2000 rise 2 fall 3

backend varnish.offer99.com
```



```
server varnish 222.35.135.152:81 weight 5 check inter 2000 rise 2 fall 3
```

```
backend www.offer99.com
    balance source
    option httpchk HEAD /index.php HTTP/1.0
    server web1 222.35.135.154:80 weight 5 check inter 2000 rise 2 fall 3
    server web2 222.35.135.155:80 weight 5 check inter 2000 rise 2 fall 3
```

下面将针对 HAProxy 配置文件的正则情况稍做说明：

```
acl cache_domain path_end .css .js .gif .png .swf .jpg .jpeg
```

以上语句的作用是将 .css 和 .js 及图片类型文件定义成 cache_domain。

```
acl cache_dir path_reg /apping
acl cache_jpg path_reg /theme
```

以上两句话的作用是定义静态页面路径，cache_dir 和 cache_jpg 是自己随便取的名字。

```
use_backend varnish.offer99.com if cache_domain
use_backend varnish.offer99.com if cache_dir
use_backend varnish.offer99.com if cache_jpg
```

如果满足以上文件后缀名或目录名，则 HAProxy 将客户端请求定向到后端的 Varnish 缓存服务器 varnish.offer99.com 上。

```
acl bugfree_domain path_reg /bugfree
use_backend bugfree.offer99.com if bugfree_domain
```

以上两句话的配置文件是将 bugfree 专门定义成一个静态域，如果客户端有 bugfree 的请求，则专门定向到后端的 222.35.135.151 机器上。

```
default_backend www.offer99.com
```

如果客户端的请求都不满足以上条件，则分发到后端的两台 Apache 服务器上。

建议将配置文件写成 Frontend（前台）和 Backend（后台）的形式，方便我们根据需求利用 HAProxy 的正则做成动静分离或根据特定的文件名后缀（比如 .php 或 .jsp）访问指定的 phppool 池或 javapool 池（其实就是 php 或 java 服务器集群）；我们还可以指定静态服务器池，让客户端访问静态文件（比如 bmp、css 或 js）时可以访问 Varnish 缓存服务器集群，这就是大家常说的动静分离功能了（Nginx 也能实现此项功能），所以前后台的模型也是非常有用的。

Keepalived 的配置过程比较简单，这里略过，大家可以参考前面的配置，配置成功后可以分别在两台机器上启动 HAProxy 及 Keepalived 服务，主机上 Keepalived.conf 的配置文件内容如下：

```
! Configuration File for keepalived
global_defs {
    notification_email {
```

```

yuhongchun027@163.com
}
notification_email_from sns-lvs@gmail.com
smtp_server 127.0.0.1
router_id LV5_DEVEL
}
vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        203.93.236.149
    }
}

```

3. 替 HAProxy 添加日志支持

编辑 /etc/syslog.conf 文件, 添加如下内容:

```

local3.*      /var/log/haproxy.log
local0.*      /var/log/haproxy.log

```

编辑 /etc/sysconfig/syslog 文件, 修改内容如下:

```
SYSLGOD_OPTIONS="-r -m 0"
```

然后重启 syslog 服务, 命令如下:

```
service syslog restart
```

在这里有一点需要说明一下, 在实际的生产环境下, 开启 HAProxy 日志功能是需要硬件成本的, 它会消耗大量的 CPU 资源, 这会导致系统速度变慢 (这点在硬件配置较弱的机器上表现尤其突出), 如果不需要开启 HAProxy 日志功能可以选择将其关闭, 大家可以根据实际需求来选择是否需要开启 HAProxy 日志。当时线上采用的机器类型为 DELL 2950, 机器的 CPU 性能有些偏弱, 上线以后我们关闭了 HAProxy 日志。

4. 验证此架构及注意事项

我们可以通过关闭或重新启动主 HAProxy 机器来测试 VIP 地址有没有被正确地转移到从 HAProxy 机器上, 以及是否影响到对网站的访问了。上面的步骤笔者测试过多次, 而且在线上环境里也已稳定运行, 证明 HAProxy+Keepalived 双机方案确实是有效的。

可能有读者会有疑问, 在此 HAProxy+Keepalived 负载均衡高可用的架构中, 我们是如何解决 Session 共享的问题呢? 答案是采用它自身的 balance source 机制, 它跟 Nginx 的 ip_hash 机制的原理类似, 是让客户机访问时始终访问后端的某一台真实的 Web 服务器, 这样

就可让 Session 固定下来了，这里没有为了节约机器成本，而采用 Memcached 或 redis 来作为 Session 共享机器。

```
option httpchk HEAD /index.php HTTP/1.0
```

这行代码的作用是进行网页监控，如果 HAProxy 检测到 Web 的根目录下不存在 index.jsp，就会产生 503 报错。

5. HAProxy 的监控页面

可以在地址栏输入 `http://www.offer99.com/haproxy-stats/`，输入用户名和密码后，显示界面如图 7-10 所示（HAProxy 自带监控页面，这也是笔者非常喜欢的功能之一）。

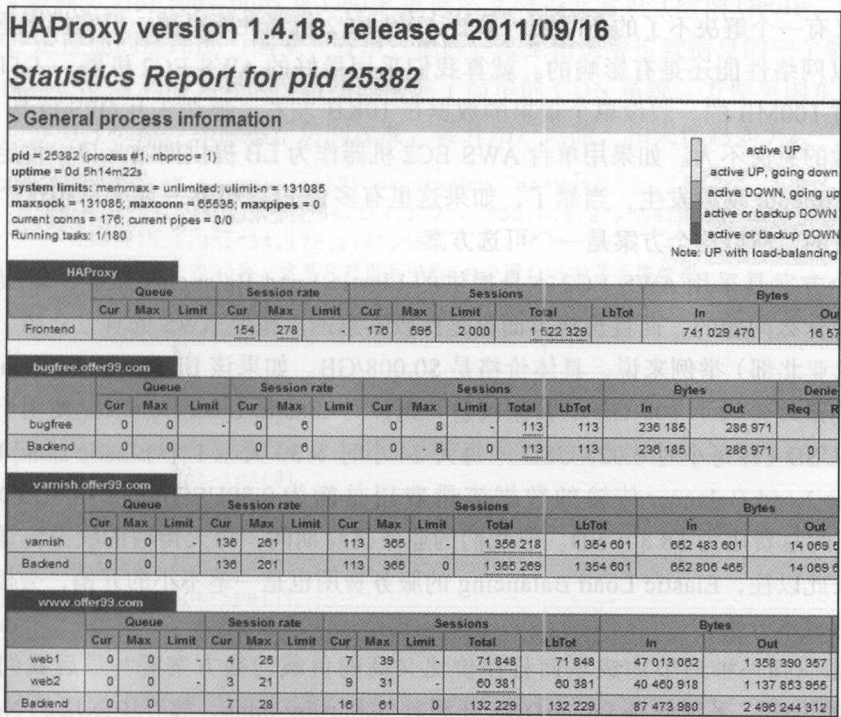


图 7-10 HAProxy 自带的监控页面

说明 “Session rate”的“Cur”选项可以反映网站的即时并发数，这是大家最关心的选项之一了，还可以利用此监控页面关注 Web 服务器的存活信息等，HAProxy 的这项功能相当实用。

此套集群方案上线以来，HAProxy 负载均衡机器运行得相当稳定，在新广告上线（高流量高并发）的情况下基本没出现过宕机现象，所以笔者也没有像 Nginx+Keepalived 那样做 HAProxy 服务级别的监控，仅仅做了双机的 Keepalived，以避免单机服务器硬件故障。如果大家想在生产环境下实施 HAProxy+Keepalived，可以参考此文档来进行部署实施。

7.4.8 案例分享八：巧用 DNS 轮询做负载均衡

笔者正在维护的 DSP 大型广告平台，用的是纯 AWS 云平台环境，业务高峰期时有十几万的并发量，3 万左右的 QPS，不仅 Web 层面，后面的数据库和 redis 缓存也面临着巨大压力（这一块的内容将在后面的章节里进行详细说明）。这么巨大的流量和并发量，只能采用分布式的思路来解决。

第一个方案是在前端用 CDN 的方式来解决压力的问题。但由于 DSP 业务的特殊性质，3 万多 QPS 请求基本上都是动态请求，而非静态图片或 CSS 等静态请求等。所以前端放置 CDN 的方案第一个被否决。

第二个方案是利用 AWS EC2 机器将其作为 HAProxy/Nginx 以起到承担分流的作用，但是这里又有一个解决不了的新问题。虽然 AWS EC2 机器性能卓越，但它们毕竟是共享带宽的，所以网络性能还是有影响的。就算我们采用最好的 AWS EC2 机器，入口带宽也是不可能超过 100MB 的。假设单个请求的数据在 10KB 左右，那么 3 万 QPS 就是 300MB/s，所以这样做的意义不大。如果用单台 AWS EC2 机器作为 LB 提供网站入口，肯定会有大量的丢包和 Timeout 现象发生，当然了，如果这里有多台 EC2 机器一起来承担这个工作，这个还是可行的，所以这个方案是一个可选方案。

第三个方案是采用 AWS EC2 本身提供的 Elastic Load Balancing 服务，就服务本身而言是完全没什么问题的，而且价格方面也是比较实惠的。按照 AWS 的官方介绍，以美国东部（弗吉尼亚北部）举例来说，具体价格是 \$0.008/GB。如果该 Elastic Load Balancer 在 30 天的期间内最终传输了 100 GB 的数据流量，则该月 Elastic Load Balancer 使用小时数费用总额为 18 USD（即每小时 0.025 USD × 每天 24 小时 × 30 天 × 1 个 Elastic Load Balancer），通过 Elastic Load Balancer 传输的数据流量费用总额为 0.80USD（即 0.008USD/GB × 100 GB），该月的总费用为 18.80USD。但我们的业务高峰期间，每天传输的数据流量远远不止 100GB。长此以往，Elastic Load Balancing 的服务费用也是一笔不小的开销，会极大地增加网站的运营成本开销。

能不能找到一种最节约成本而且性价比又高的负载均衡方案呢？最后我们想到了用 DNS 轮询的方案，采用的是 PowerDNS 开源软件和 ruby-pdns。放弃中间层的 Nginx（LB）机器，直接将 DNS 指向为后端的 bidder 机器。

PowerDNS 是高性能的域名服务器，除了支持普通的 bind 配置文件，PowerDNS 还可以从 MySQL、Oracle、PostgreSQL 等数据库中读取数据。PowerDNS 安装了 Poweradmin，能实现 Web 管理 DNS 记录，非常方便。

ruby-pdns 是一个简单的 Ruby 库，可用来开发基于 PowerDNS 的 DNS 动态记录应用，它将复杂的 DNS 操作过程封装起来并提供简单易用的方法，示例代码如下所示。

```
module Pdns
  newrecord("www.your.net") do |query, answer|
    case country(query[:remoteip])
```



```

when "US", "CA"
  answer.content "64.xx.xx.245"
when "ZA", "ZW"
  answer.content "196.xx.xx.10"
else
  answer.content "78.xx.xx.140"
end
end
end

```

工作中采用 ruby-pdns 的主要原因是：修改 PowerDNS 记录是即时生效的，无须重启 PowerDNS 服务。另外 ruby-pdns 对 GeoIP 数据库支持得非常好（所谓 GeoIP，就是通过来访者的 IP，定位他的经纬度、国家、省市、地区，甚至街道等位置信息的一个数据库）。在此业务系统中，笔者利用其智能解析功能搭建了简单的 CDN 系统，方便美国东西部客户就近连结其业务图片机器，加快用户访问速度，提升用户体验，相关代码如下：

```

newrecord("bid-east.example.net") do |query, answer|
  ips = ["54.175.1.2", "54.164.1.2", "52.6.1.2", "54.164.1.2", "54.175.1.2",
    "54.175.1.3", "54.175.1.4", "52.4.1.2"..... ]
  #bidder机器大约20台，这里只列出其中的8台公网IP并做了无害处理
  ips = ips.randomize([1, 1, 1, 1, 1, 1, 1, 1])
  answer.shuffle false
  answer.ttl 30
  answer.content ips[0]
  answer.content ips[1]
  answer.content ips[2]
  answer.content ips[3]
  answer.content ips[4]
  answer.content ips[5]
  answer.content ips[6]
  answer.content ips[7]
end

module Pdns
  newrecord("ads.bilinmedia.net") do |query, answer|
    country_, region_ = country(query[:remoteip])
    answer.qclass query[:qclass]
    answer.qtype :A
    case country_
      when "US"
        case region_
          when "WI", "IL", "TN", "MS", "ID", "KY", "AL", "OH", "WV",
            "VA", "NC", "SC", "GA", "FL", "NY", "PA", "ME", "VT",
            "NH", "MA", "RI", "CT", "NJ", "DE", "MD", "DC"
            # 东部地区用户访问东部图片服务器
            answer.ttl 300
            answer.content "54.165.1.2"

```



```

else
    # 西部地区用户访问西部图片服务器
    answer.ttl 300
    answer.content "54.67.1.2"
end
else
    # 如果用户IP都不在上面的城市，则选择默认的西部机器
    answer.ttl 300
    answer.content "54.67.1.2"
end
end
end
end

```

DNS 轮询主要是靠如下代码来实现的。

```

newrecord("bid-east.example.net") do |query, answer|
    ips = ["54.175.1.2", "54.164.1.2", "52.6.1.2", "54.164.1.2", "54.175.1.2", "
54.175.1.3", "54.175.1.4", "52.4.1.2"..... ]
    #bidder机器大约20台，这里只列出其中的8台，公网IP做了无害处理
    ips = ips.randomize([1, 1, 1, 1, 1, 1, 1, 1])
    answer.shuffle false
    answer.ttl 30
    answer.content ips[0]
    answer.content ips[1]
    answer.content ips[2]
    answer.content ips[3]
    answer.content ips[4]
    answer.content ips[5]
    answer.content ips[6]
    answer.content ips[7]
end
end

```

我们可以用 dig 命令来解析下 bid-east.example.com 域，命令如下所示：

```
dig bid-east.example.com
```

命令结果显示如下所示：

```

; <<>> DiG 9.3.6-P1-RedHat-9.3.6-20.P1.el5_8.6 <<>> bid-east.bilinmedia.net
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 36017
;; flags: qr rd ra; QUERY: 1, ANSWER: 8, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;bid-east.bilinmedia.net. IN      A

;; ANSWER SECTION:
bid-east.bilinmedia.net. 30      IN      A      54.175.1.2
bid-east.bilinmedia.net. 30      IN      A      54.164.1.2
bid-east.bilinmedia.net. 30      IN      A      52.6.1.2
bid-east.bilinmedia.net. 30      IN      A      54.164.1.2

```

```

bid-east.bilinmedia.net. 30      IN      A       54.175.1.2
bid-east.bilinmedia.net. 30      IN      A       54.175.1.3
bid-east.bilinmedia.net. 30      IN      A       54.175.1.4
bid-east.bilinmedia.net. 30      IN      A       52.4.1.2
.....

;; AUTHORITY SECTION:
bid-east.bilinmedia.net. 1799    IN      NS     ns.bilinmedia.net.

;; ADDITIONAL SECTION:
ns.bilinmedia.net. 599          IN      A       54.173.66.112

;; Query time: 1530 msec
;; SERVER: 10.143.22.116#53(10.143.22.116)
;; WHEN: Thu Jan 14 09:55:23 2016
;; MSG SIZE rcvd: 202

```

这样配置上去以后，在业务最繁忙的时间段观察可以得知：20台 bidder 机器，Nginx+Lua 作为 Web 服务器，平均每台的活动连接数在 20 000 ~ 22 000 左右，流量被平均分担下去了，达到了负载均衡的目的。我们可以用 Ansible 工具抽取下空闲时间（晚上凌晨 2 点左右）bidder 集群机器的活动连接数情况，命令如下所示：

```
ansible bidder -m script -a "/home/ec2-user/counter.sh"
```

结果如下所示：

```

bidder1 | SUCCESS => {
  "changed": true,
  "rc": 0,
  "stderr": "",
  "stdout": "FIN_WAIT1 13,ESTABLISHED 3193,LISTEN 6\r\n",
  "stdout_lines": [
    "FIN_WAIT1 13,ESTABLISHED 3193,LISTEN 6"
  ]
}
bidder2 | SUCCESS => {
  "changed": true,
  "rc": 0,
  "stderr": "",
  "stdout": "TIME_WAIT 1,FIN_WAIT1 9,ESTABLISHED 3175,SYN_RECV 2,LISTEN 8\r\n",
  "stdout_lines": [
    "TIME_WAIT 1,FIN_WAIT1 9,ESTABLISHED 3175,SYN_RECV 2,LISTEN 8"
  ]
}
bidder4 | SUCCESS => {
  "changed": true,
  "rc": 0,
  "stderr": "",
  "stdout": "FIN_WAIT1 15,ESTABLISHED 3176,LISTEN 6\r\n",
  "stdout_lines": [

```

```

        "FIN_WAIT1 15,ESTABLISHED 3176,LISTEN 6"
    ]
}
bidder5 | SUCCESS => {
    "changed": true,
    "rc": 0,
    "stderr": "",
    "stdout": "TIME_WAIT 1,FIN_WAIT1 10,ESTABLISHED 3262,LISTEN 6\r\n",
    "stdout_lines": [
        "TIME_WAIT 1,FIN_WAIT1 10,ESTABLISHED 3262,LISTEN 6"
    ]
}
bidder3 | SUCCESS => {
    "changed": true,
    "rc": 0,
    "stderr": "",
    "stdout": "TIME_WAIT 2,FIN_WAIT1 15,ESTABLISHED 3857,LISTEN 6\r\n",
    "stdout_lines": [
        "TIME_WAIT 2,FIN_WAIT1 15,ESTABLISHED 3857,LISTEN 6"
    ]
}
bidder7 | SUCCESS => {
    "changed": true,
    "rc": 0,
    "stderr": "",
    "stdout": "FIN_WAIT1 7,ESTABLISHED 2821,LISTEN 6\r\n",
    "stdout_lines": [
        "FIN_WAIT1 7,ESTABLISHED 2821,LISTEN 6"
    ]
}
bidder6 | SUCCESS => {
    "changed": true,
    "rc": 0,
    "stderr": "",
    "stdout": "TIME_WAIT 1,FIN_WAIT1 8,ESTABLISHED 3239,LISTEN 6\r\n",
    "stdout_lines": [
        "TIME_WAIT 1,FIN_WAIT1 8,ESTABLISHED 3239,LISTEN 6"
    ]
}
bidder8 | SUCCESS => {
    "changed": true,
    "rc": 0,
    "stderr": "",
    "stdout": "TIME_WAIT 1,FIN_WAIT1 7,ESTABLISHED 3238,LISTEN 6\r\n",
    "stdout_lines": [
        "TIME_WAIT 1,FIN_WAIT1 7,ESTABLISHED 3238,LISTEN 6"
    ]
}
}

```

基本上, Nginx 的活动并发连接数也是比较平均的, 维持在 3 200~3 800 左右, 证明流

量是平均分配下来的，PowerDNS 的轮询功能是生效的，业务繁忙的时候通过 ruby-pdns 修改其配置文件，动态地添加 bidder 业务机器，就可以很轻松地进行水平扩展以应付新增的流量了。

7.5 软件级负载均衡器的特点介绍与对比

现在网站发展的趋势对网络负载均衡的要求是：随着网站规模的提升，根据不同的阶段使用不同的技术，如下所示：

一种是通过硬件来进行，常见的硬件有比较昂贵的 NetScaler、F5 Big-IP 等商用的负载均衡器，它的优点就是性能稳定、模块丰富，并且有专业的维护团队来做维护，缺点就是花销太大，所以对于规模较小的网络服务来说暂时还没有使用它的必要；另外一种就是类似于 LVS/HAProxy、Nginx 的、基于 Linux 的负载均衡软件策略，这些都是通过软件来实现的，所以费用非常低廉，故而推荐大家采用第二种方案来实施自己网站的负载均衡需求。

很多朋友担心软件级别的负载均衡在高并发流量冲击下的稳定情况，事实是我们通过成功上线的许多网站和系统可以发现，软件级别的负载均衡的稳定性也是非常好的，宕机的可能性微乎其微，下面就它们的特点和适用场合分别进行说明。

1. LVS

它是使用集群技术和 Linux 操作系统实现的一个高性能、高可用服务器，具有很好的可伸缩性（Scalability）、可靠性（Reliability）和可管理性（Manageability），感谢章文嵩博士为我们提供如此强大实用的开源软件。

LVS 的特点是：

- ❑ 抗负载能力强、工作在网络四层之上仅作分发之用，DR 模式没有流量的产生，这个特点也决定了它在负载均衡软件里的性能是最强的。
- ❑ 配置性比较低，这是一个缺点也是一个优点，因为没有太多可配置的东西，所以并不需要太多接触，大大减少了人为出错的概率。
- ❑ 运行稳定，自身有完整的双机热备方案，如 LVS+Keepalived 和 LVS+Heartbeat，不过在项目实施中用得最多的还是 LVS/DR+Keepalived，建议大家也多关注下淘宝开发的 LVS/FullNAT 模式。
- ❑ 无流量，保证了均衡器 IO 的性能不会受到大流量的影响。
- ❑ 应用范围比较广，可以对所有应用做负载均衡。
- ❑ 软件本身不支持正则处理，不能做动静分离，这个就比较遗憾了；其实现在许多网站在这方面都有较强的需求，这个是 Nginx/HAProxy+Keepalived 的优势所在。

- ❑ 如果网站应用比较庞大，实施 LVS/DR+Keepalived 就会比较复杂，特别是如果后面有 Windows Server 应用的机器的话，实施、配置及维护过程就会更复杂了，相对而言，Nginx/HAProxy+Keepalived 的部署及维护就简单得多了。

2. Nginx

Nginx 的特点是：

- ❑ 工作在网络的七层之上，可以针对 HTTP 应用做一些分流的策略，比如针对域名、目录结构，它的正则规则比 HAProxy 更为强大和灵活，这也是许多朋友喜欢它的原因之一。
- ❑ Nginx 对网络的依赖性非常小，理论上能 ping 通就能使用 Nginx 进行负载功能，这个也是它的优势所在。
- ❑ Nginx 的安装和配置比较简单，测试起来比较方便。
- ❑ Nginx 既可以承担很高的负载压力又很稳定，一般能支撑超过几万次的并发量。
- ❑ Nginx 可以通过端口检测到服务器内部的故障，比如根据服务器处理网页返回的状态码、超时等，并且会把返回错误的请求重新提交到另一个节点，不过其中的缺点就是不支持 URL 来检测。
- ❑ Nginx 只能支持 HTTP 和 E-mail，这样其适用范围就小了很多，这个是它的弱势。
- ❑ Nginx 不仅仅是一款优秀的负载均衡器 / 反向代理软件，同时也是功能强大的 Web 应用服务器。LNMP 现在也是非常流行的 Web 架构，大有和以前最流行的 LAMP 架构分庭抗礼之势，在高流量的环境中也有很好的效果。
- ❑ 如今，Nginx 作为 Web 反向加速缓存变得越来越成熟了，很多朋友都已将其投入生产环境中进行生产了，而且反映效果还不错，速度比传统的 Squid 服务器更快，有此需求的朋友可以考虑在其合适的工作场景中将其作为反向代理加速器。

3. HAProxy

HAProxy 的特点是：

- ❑ 抗负载能力强，兼备四层和七层负载均衡的作用，可以代替 LVS，用于四层负载均衡分发流量之用。
- ❑ 支持虚拟主机。
- ❑ 能够补充 Nginx 的一些缺点比如 Session 的保持，Cookie 的引导等工作。
- ❑ 跟 LVS 一样，本身仅仅只是一款负载均衡软件；单纯从效率上来讲，HAProxy 比 Nginx 有更出色的负载均衡速度，在并发处理上也是优于 Nginx 的。

7.6 网站系统架构设计图

笔者工作中的部分网站系统架构设计图，分别如图 7-11 至图 7-14 所示：

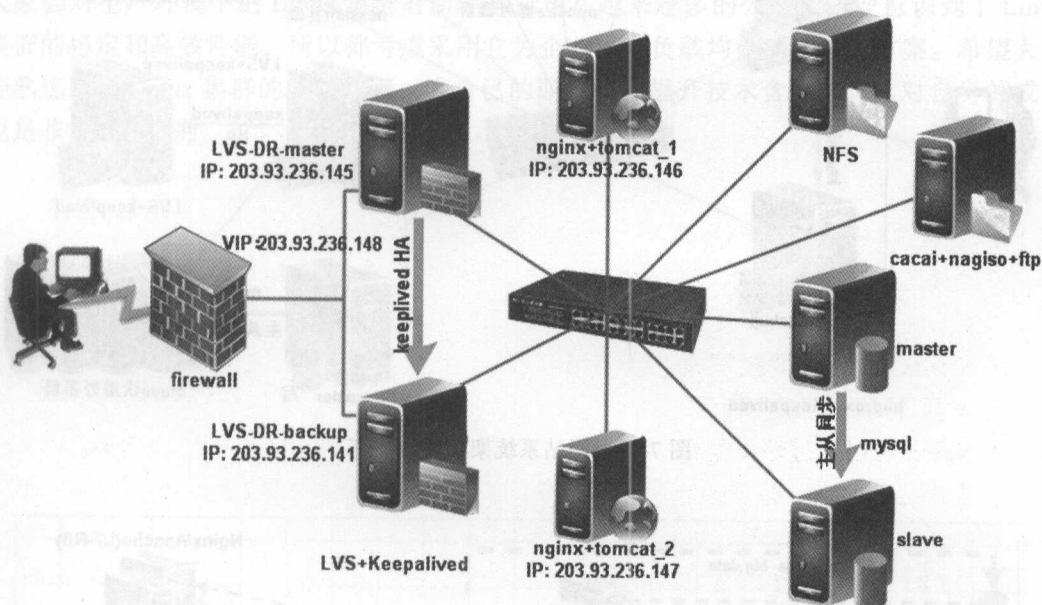


图 7-11 网站系统架构设计图一

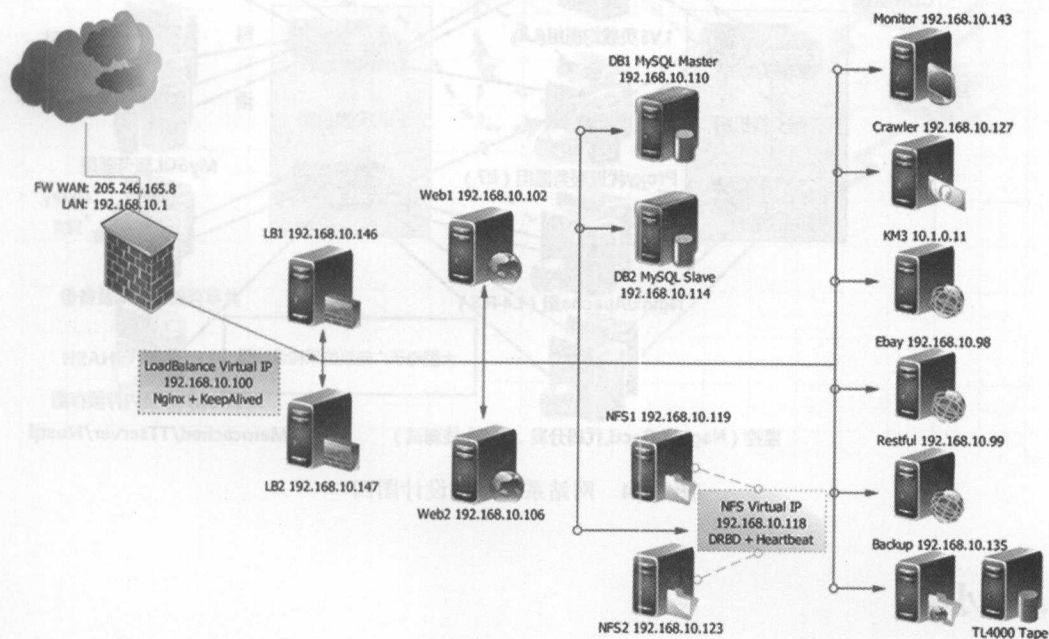


图 7-12 网站系统架构设计图二

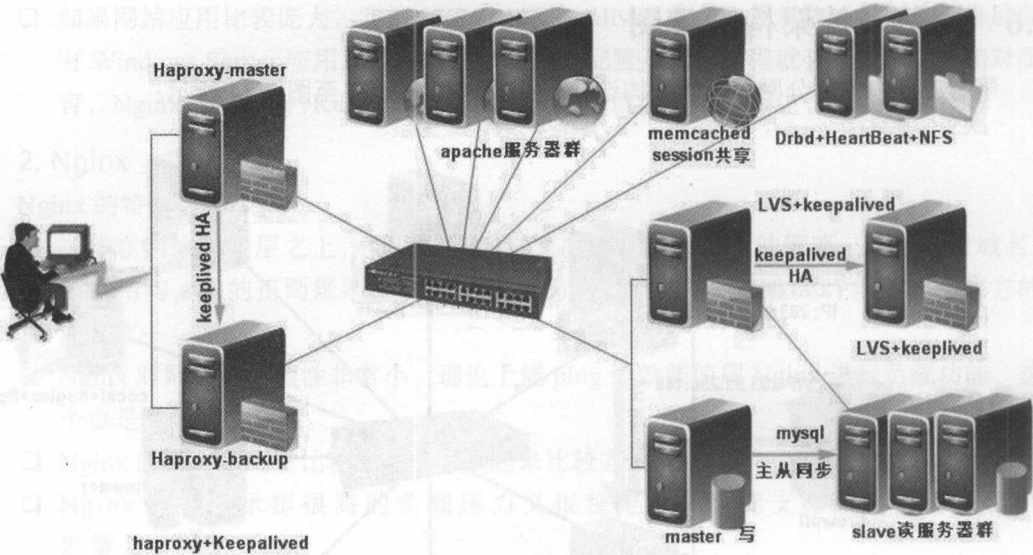


图 7-13 网站系统架构设计图三

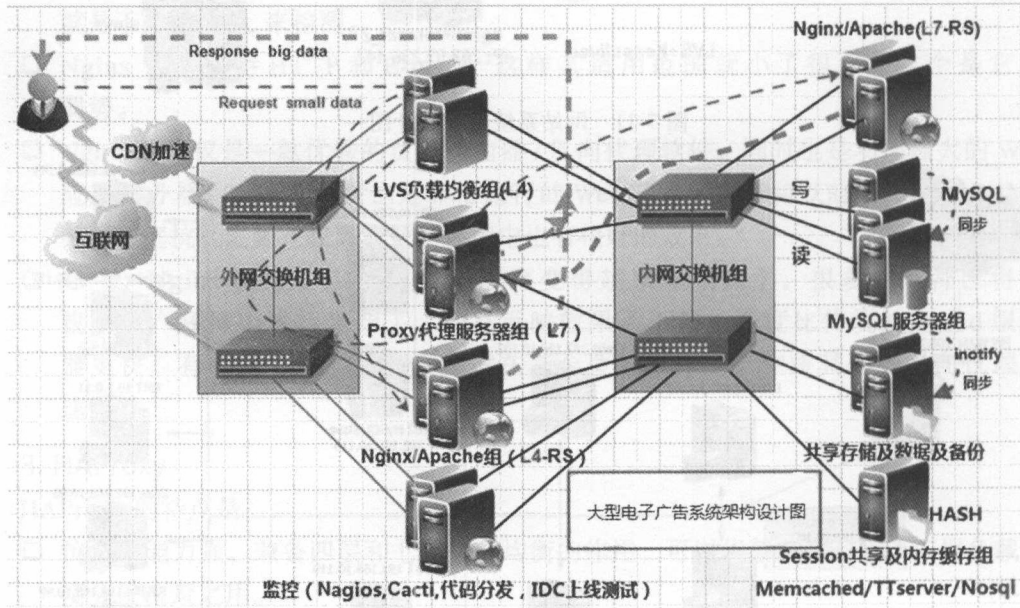


图 7-14 网站系统架构设计图四

7.7 小结

这一章主要向大家介绍了Linux 集群技术所采用的开源软件，例如LVS、HAProxy、

Nginx 及 DNS 轮询等,并介绍了负载均衡中的常用技术,比如:Session 共享、会话保持、常见的算法等。另外还通过真实的项目演示着重介绍了现在比较流行 Nginx/HAProxy+Keepalived、DRBD+Heartbeat 等负载均衡高可用技术,相信通过阅读这节内容,大家会对生产环境下的 Linux 集群有所了解。现在越来越多的公司和企业意识到了 Linux 集群的稳定和高效性能,所以都考虑采用它为企业提供负载均衡高可用的方案。希望大家能熟练掌握 Linux 集群的相关知识,为自己的职业技能提升技术含金量,这对自身的成长也是非常有帮助的。

8.1.3 合理选用开源软件方案

现在的硬件虽然性能卓越,稳定性也很强,但价格确实比较昂贵,从长期运营的角度考虑,如果选择以硬件为基础的架构,那么随着业务量的增加,成本也需要随之增加,而开源软件则不同,开源软件的成本相对较低,而且可以随着业务量的增加而进行灵活的扩展。另外,开源软件通常具有更好的兼容性和可扩展性,可以更好地满足企业的需求。因此,在构建 Linux 集群时,合理选用开源软件方案是一个明智的选择。常见的开源软件方案包括 Nginx、HAProxy、Keepalived、DRBD、Heartbeat 等,这些软件方案在业界得到了广泛的应用和认可。

Oracle RAC 商用集群价格不菲,可以考虑使用开源的集群方案,比如 MySQL 集群、Redis 集群等,这些方案在性能和稳定性上都能满足大部分业务需求,而且价格相对较低。

另外,在数据库选型上,除了 MySQL 和 Redis 之外,还可以考虑使用其他开源数据库,比如 PostgreSQL、MongoDB 等。在选择数据库时,需要根据业务需求进行综合考虑,比如数据量、读写性能、高可用性等。同时,在选择开源软件方案时,还需要关注社区的支持和活跃度,以确保软件能够及时得到维护和更新。

8.1.4 机房及 CDN 选型

机房的选型比较重要,我们到底是选择自建机房还是租用第三方机房,需要根据业务实际情况来决定。自建机房的优势在于可以更好地控制硬件和软件环境,但成本较高;租用第三方机房则可以节省成本,但需要关注机房的安全性和稳定性。

如果业务系统的小程序较多,为了提升用户体验,可以考虑使用 CDN 加速。CDN 可以将静态资源分发到全球各地的边缘节点,从而加快用户访问速度。在选择 CDN 服务时,需要关注服务商的覆盖范围、带宽、价格等因素。

在用户体感的角度,建议大家采用 BGP 线路,至少也要采用多条线路,以确保用户能够访问到最近的节点,提升访问速度。

浅谈网站系统架构设计

作为一名运维架构师，很多时候需要设计公司的电子商务或业务网站的系统架构，这个时候就需要根据业务需求及公司自身的实际情况，以预算为前提，设计出一套高可用、高性能、高可扩展性的架构预案并评估其实际可实施性。正式启动此架构预案方案的时候，运维架构师也必须要亲力亲为地实施、维护并进行合理优化，保证公司的电子商务网站或其他业务网站的稳定运行，这个是运维架构师的职责所在。

8.1 网站架构设计规划预案

8.1.1 利用经验，合理设计

运维架构师应根据之前的工作经验，设计合理的网站架构方案，引导技术团队树立正确的系统架构设计思想，指明正确的方向，规避以后网站升级可能会存在的风险，具体如下所示：

- ❑ 高流量高并发的网站一定要采用分布式的架构思想来设计，可以采用 DNS 轮询或 LVS 负载均衡将最外面的流量进行一级分流。
- ❑ 合理利用 CDN 系统，注意 CDN 回源的问题。
- ❑ 图片服务器采用独立域名，而非二级域名。
- ❑ 尽量选用 BGP 机房或线路。
- ❑ 成本方面一定要控制，尽量选用免费开源方案。
- ❑ 尽量选用目前比较成熟稳定的技术。
- ❑ 慎重选用业务核心系统的开发语言和开发框架，在后期进行代码重构将是一件非常复杂和痛苦的事情。

8.1.2 规划好网站未来的发展

在设计网站的系统架构之初,要评估下当前系统的 PV、UV 及 QPS,做好系统上线以后未来 3 到 5 年的数据扩展方案,尽量将网站做成高可扩展性的,这样方便以后进行横向扩展(Scale Out,也叫水平扩展,纵向扩展是通过增添机器硬件的方式进行扩展),说白了就是能够很方便地增添新机器。此外,还需要考虑如下因素:

- ❑ 重要的对外服务尽量做成集群的形式,这样可避免因单机宕机而中断服务。
- ❑ 存放数据的磁盘类型可以考虑用 LVM 逻辑卷,以方便扩容。
- ❑ 机房需要考虑是否异地冗余,以及在核心机房出问题能否快速切换。

另外,在业务高峰期,PV、UV 及 QPS 迅速增加、增大的时候,集群能否马上提供应急机器;而在非业务高峰期间,因为应急而上线的机器又该如何平稳地下线?这些都是运维架构师要慎重考虑的地方。

8.1.3 合理选用开源软件方案

现在的硬件虽然性能卓越,稳定性也很强,但价格确实比较昂贵,从长期运营的角度来考虑,开源软件方案应该是我们的首选。

负载均衡设备 NetScaler 和 F5 的价格都比较贵,因此最好合理选用免费、开源的 LVS/Nginx/HAProxy 方案。

IBM 的服务器性能确实强悍,但价格昂贵。事实上,自前端有了负载均衡设备以后,廉价的服务器甚至 PC 机器都可以为我们提供稳定的服务。

EMC 的 APP 高端存储确实不错,但价格实在是太昂贵了,可以考虑分布式文件 MooseFS 或 NFS 分组。

Oracle RAC 商用集群价格不菲,可以考虑用免费的 MySQL 来设计数据库架构,一主多从、读写分离是不错的选择。

Varnish 和 redis 这些新起的缓存软件虽然是开源免费的,但性能方面并不亚于商业软件,完全可以考虑在自己的网站中合理地利用它们。

至于海量离线日志处理系统,可以考虑开源的 Hadoop/Spark 集群。

海量内容搜索引擎,可以考虑开源的 ElasticSearch。

8.1.4 机房及 CDN 选型

机房的选型比较重要,我们到底是选择机房托管还是采用云计算平台呢?这个要根据业务的实际情况来确定了。

如果业务系统的小图片过多,为了加快用户的访问速度,并提升用户体验,建议大家租用 CDN,这里不推荐自建 CDN 的方式。因为这样做,成本和性能不成正比,除非是要提供专业的视频或图片服务的网站。

站在用户体验的角度,建议大家采用 BGP 机房,至少也要选用双线机房。

可以根据业务选型来选择机房，如下所示：

- ❑ 如果是提供资讯类的网站，建议大家采用机房托管的方式，自建机房的成本较高。
- ❑ 如果公司是专业的电子商务网站，牵涉到在线交易系统的话，建议大家选用机房托管的方式，如果自己有机房的话那就最好了，核心数据机器要放在自己能掌控的范围之内。
- ❑ 如果是 CPA 或 DSP 等广告系统的话，由于牵涉多地区布点的问题，建议采用云计算平台的方式，推荐亚马逊云。
- ❑ 如果仅仅只是自己的门户网站，从性价比的角度考虑，推荐阿里云。

8.1.5 节约成本

网站上线运行之后，运行成本、带宽费用、机器升级和维护费用均不便宜，所以我们要在网站上线之前合理规划，节约其成本，这里拿 AWS 云计算平台提供的产品来举例说明。

对于 AWS EC2 机器，如果是 On Demand（按需使用）用途的，大家尽量以预留实例的方式运行，毕竟预留实例跟实例的价格相差 40% ~ 50%。

此外，要合理利用 Spot Instance。考虑到服务器资源在各个时段和地区的利用率都是不一样的，如果有利用率低的情况，AWS 就可以把空闲的服务器资源以一个很低的价格提供给对地区或时间并不敏感的应用。比如我要运行分布式爬虫程序去爬一些数据，实际运行时间可能也就几个小时，这个时候就可以考虑采用 Spot Instance。考虑到 Spot Instance 的价格往往低至常规的 On Demand 价格的 1/10，因此灵活使用 Spot Instance 可以节省大量预算。

亚马逊默认是按照 CPU 的个数来收取费用的，其收费标准跟 CPU 个数成正比，所以我们应尽可能地在非计算类型的机器上控制 CPU 的数量。

S3（Simple Storage Service）文件系统能为我们提供高可用和可信存储，但价格比较昂贵，我们可以考虑用 Amazon Glacier 来存放历史数据以降低使用 S3 的费用。Glacier 是成本极低的存储服务，为数据备份和存档提供了安全、耐用、灵活的存储。

8.1.6 安全备份

在安全方面，需要考虑以下几个问题：

- ❑ 需要硬件防火墙吗？有没有应对 DDoS 的措施？
- ❑ 如果是金融类网站，如何保证支付安全，多域名的 HTTPS 支持呢？
- ❑ 核心数据有没有考虑异地容灾备份？
- ❑ 如何保证代码的安全，如何控制项目组成员的访问权限。
- ❑ 如果提交到线上的代码有重大 Bug，那么如何回滚代码对业务的影响才会最小。

综上所述，网站架构设计需要考虑的地方非常多，要全面规避风险，需要运维架构师有足够的经验，根据网站冗余及可扩展性，进行异地容灾备份，控制成本等，针对自己的

公司业务设计一个最优的配置方案。

8.2 百万级 PV 高可用网站架构设计

在许多小公司和小企业，尤其是涉及电子广告和电子资讯类的网站，其网站的日 PV 不超过一百万，但由于其重要性，也要求网站应该具备负载均衡高可用的特点；另一方面，由于成本的制约，公司都会要求系统架构师设计的方案能够用最少的预算来满足这个要求，作为运维架构师的我们，应该如何满足这个要求呢？

首先是机房的选择，如果公司有自己的机房那是最好不过的了；如果没有自己的机房，建议放在 BGP 机房内托管，最好是选择带有硬件防火墙的 BGP 机房，在安全方面会更有保障。另外，如何选择服务器呢？在有了负载均衡高可用的集群环境后，我们完全可以自己组装服务器，这在性价比上也是最高的。像 IBM 和 DELL 等品牌的服务器，虽然质量有保障，但价格还是比较昂贵的。当然了，一切以稳定为前提和原则。

另外，如果考虑用云产品来部署自己的网站，可以对比下阿里云和亚马逊云的价格，亚马逊 AWS 宣布入华后，阿里云的全线产品下降了 30%。而云计算相比传统 IT 的最大优势在于成本。如果对比下同等规模的机器，可以发现阿里云在价格上具有绝对的优势。所以对于小型网站来说，可以考虑采用阿里云的方式进行部署。

网站架构设计首先要考虑的是负载均衡设备的选择。这里有两种选择，一种是通过硬件来进行，常见的硬件有比较昂贵的 NetScaler、F5 BIG-IP 等商用的负载均衡器，优点就是有专业的维护团队来对这些服务进行后期维护；缺点就是开销太大，所以对于规模较小的网络服务来说暂时还没有使用的需要。另外一种就是类似于 LVS/HAProxy、Nginx 等基于 Linux 的负载均衡软件策略，这些都是通过软件级别来实现的，所以费用非常低廉，小型企业和公司大都会选择软件级别的负载均衡。

至于负载均衡高可用架构，首推是 Nginx/HAProxy+Keepalived 架构，可能有读者会有疑问，为什么不选择基于 LVS/DR+Keepalived 的集群方案呢？这是因为我们部署的网站一般都会有动静分离、正则分发的需求，如果最初始选用 LVS+Keepalived 的架构，那么至少又要在中间加一层二级负载均衡的机器，这样会比较耗机器，无形中也会增加整个网站的成本。另外，很多朋友都比较担心的一个问题，认为 Nginx/HAProxy+Keepalived 的稳定性不如 LVS+Keepalived，这其实是个误解。我们通过十几个项目的实施和几年的观察，发现这些软件级别的负载均衡器的稳定性确实很好，在高并发的情况下宕机的可能性微乎其微。而近期实施的一个商业网站，用的就是 HAProxy+Keepalived，在亿 PV/日高并发流量的冲击下，HAProxy 稳如磐石。而小公司的并发和流量一般不是特别大，大概一天持续在 100 万~500 万 PV/日之间，所以这里也向大家推荐使用 HAProxy+Keepalived。

在实际的项目实施过程中笔者发现，像这样的集群方案也比较耗费资源，特别是对于网站规模较小，机器非常少的情况，效果会不太好。笔者之前维护的公司有一个新闻类网

站 <http://www.3159.com> (此域名做了无害处理, 非真实域名), 其服务器就比较少, 而且是阿里云主机, 前期做的是 HAProxy+Keepalived 集群方案, 但发现效果并不是特别好, 因为阿里云主机的带宽都是共享 100MB, 实际上分配到网站入口的带宽仅仅只有 10MB 左右, 繁忙的时候会严重影响业务, 虽然每个月我们通过加钱的方式来增加 HAProxy+Keepalived 的入口带宽, 但并不是完美的解决方案, 其他机器的共享带宽没有得到充分的利用。此外, 由于所有的机器都是独享型主机, 二层交换机也并不集中在一个机柜上, 若在中间加一层 HAProxy 代理, 网站的速度很明显又会变慢。这时候, 笔者突然想到了最简单和原始的负载均衡机制, 即 DNS 轮询, 通过在美国的 HostMonster (DNS 提供商, 国内像新网和万网均提供 DNS 轮询功能) 上配置 3 个 www 的 A 主机, 这样就解决了上面所说的一系列问题, 后期如果流量持续增加, 还可以增加机器, 将网站入口由原先的单一入口模式改成分布式的, 而且完全不影响网站的访问速度。当然了, 平时一定要注意服务器的监控和服务器的稳定情况, 毕竟每宕机一台服务器, 肯定会有用户受到影响的。

通过这次成功的项目实施, 我也明白了一个道理: 集群的部署不应该一成不变, 而应该根据具体情况具体分析, 哪种方案实用就用哪种, 哪怕是最简单的 DNS 轮询。

如果网站是放在 IDC 机房托管, 而机房的最前面也没有硬件防火墙防护, 那么应该尽量做好流量监控的工作, 笔者一般会在主 Nginx/HAProxy 上安装 Nload 软件来对流量进行监控, Nload 可以对流量进行即时监控。

很多对集群感兴趣的读者经常问我, 如果网站要部署负载均衡高可用的 Linux 集群方案, 而公司又想用最节省成本的方式来实施的话, 一般需要几台服务器呢? 如果资金比较充裕, 推荐大家用 7 台来实施, 即 LB (2 台) + Web (2 台) + MySQL (2 台) + NFS (1 台); 如果资金非常不充裕, 这个方案其实还是可以压缩的, 即 2+2 架构, 最前面是 2 台 Nginx/HAProxy+Keepalived 机器, 后面是 2 台配置比较好的 Web 机器 (推荐 DELL R710), MySQL 数据库采用一主一从的方式, 分别放在 2 台 Web 机器上, 监控的 Nagios 部署在从 Nginx/HAProxy 机器上, 流量监控一般放在主 Nginx/HAProxy 上, 软件采用的是 MRTG+Nload 的方式, 文件服务器这里用的是单 NFS, 放在备 HAProxy 机器上, Web 机器采用挂载 NFS 的目录作为本地的代码或图片存放的方法; 当然了, 如果大家的公司对文件服务器有更高要求的时候 (比如网站的图片数量比较多的时候), 可以考虑再增加一台图片服务器。

在类似以上的小公司集群架构里, 应如何解决 Session 同步的问题呢? 可以采用 Nginx 的 ip_hash 和 HAProxy 的 balance source 算法, 它们算法的原理是一样的, 都会让某一客户机在相当长的一段时间内只访问固定的后端的某台真实的 Web 服务器。这样 Session 会话就会得以保持, 我们在网站的页面上进行登录的时候, 就不会在两台 Web 服务器之间跳来跳去了, 自然也不会出现登录一次后网站又提醒你重新登录的情况, 事实上, 在千万级 PV/ 日的网站上我们也尝试过用这些方式来解决 Session 同步的问题, 效果也是相当不错的。

另外, 小公司的 Web 服务器也至少有两种选择: 一种是 Apache, 另一种是 Nginx。在流量和并发量不大的环境下, 完全可以选择 Apache 作为我们的 Web 服务器, 虽然它的抗

并发能力不高,但它的稳定性是最好的,笔者的许多电子商务网站都是基于 Apache 来提供 Web 应用的。

MySQL 在这里用的就是一主一从的设计,虽然很多朋友觉得这种设计比较简单,但事实证明,它也是最稳定的。我的电子商务网站采用的也是这种架构,几年下来,从来没有因为数据库的故障而发生丢单现象。另外,从 MySQL 机器并非仅仅只起一个备份和备机的作用,我们设计的数据库读写分离,可将后台的复杂查询转到从 MySQL 机器上以减轻主 MySQL 数据库的压力。当然了,MySQL 的主从复制状态监控也是非常重要的,笔者一般是通过 Nagios 和 Shell 脚本进行双监控的方式。

如何能帮助企业节约和省钱,这其实也是运维架构师的工作职责之一,希望大家在工作中能领悟到这点。这样设计出来的网站,极具性价比,同时又具备高可用的特点,特别适合流量不大,但稳定性要求比较高的网站,有需求的朋友可以参考此架构设计。

8.3 千万级 PV 高性能高并发网站架构设计

随着网站的知名度和宣传力度越来越高,注册用户超过千万了,而且每天都有持续上涨的趋势,而 PV/日已经有向千万/日靠近的趋势,原有的 Web 架构越来越满足不了我们的需求了。所以这时候需要设计出高性能高可用的网站架构,在这套架构里,运维架构师应该做的是提升站点的整体性能和可用性,不只是前端代理,后端应用服务器、数据库、中间件等,都要综合考虑。这个架构里任何一个点存在瓶颈,整体系统处理能力都会大打折扣,我们不要让它们之一形成短板效应,网站拓扑图如图 8-1 所示。

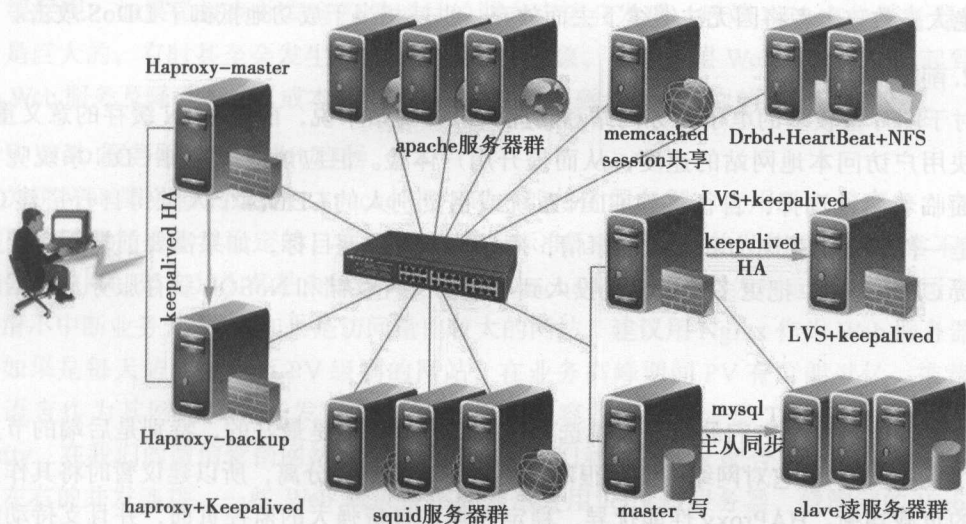


图 8-1 网站系统架构设计图

机房的选择应尽量选择 BGP 机房，双线次之。BGP 机房的优势如下：

- ❑ 服务器只需要设置一个 IP 地址，最佳访问路由是由网络上的骨干路由器根据路由跳数与其他技术指标来确定的，不会占用服务器的任何系统资源。服务器的上行路由与下行路由都能选择最优的路径，所以能真正实现单 IP 高速访问。
- ❑ 由于 BGP 协议本身具有冗余备份、消除环路的特点，所以当 IDC 服务商有多条 BGP 互联线路时可以实现路由的相互备份，其中一条线路出现故障时路由会自动切换到其他线路。
- ❑ 使用 BGP 协议还可以使网络具有很强的可扩展性，可以将 IDC 网络与其他运营商互联，轻松实现单 IP 多线路，使得所有互联运营商的用户访问都很快，这个是双 IP 双线路无法比拟的。

1. 硬件防火墙（可选）

硬件防火墙的模式可以选择路由和透明两种，可根据具体环境而定。防火墙的型号一般选择用华赛或 Juniper，大家可以根据自己业务网站的实际需求来加以选择，硬件防火墙的主要作用是用来防止 DDoS 攻击和端口映射。当然了，因为现在网站基本上都会有 CDN 服务，是否增加硬件防火墙是可以考虑的。

如果我们的网站是用于电子商务支付系统的，建议在前端放置硬件防火墙，国内的 DDoS 攻击是非常流行的，对付 DDoS 是一个复杂而且庞大的系统工程，想仅仅依靠某种系统或产品来防住 DDoS 是不现实的。可以肯定的是，目前完全杜绝 DDoS 是不可能的，但通过适当的措施抵御 90% 的 DDoS 攻击是可以做到的，基于攻击和防御都有成本开销的缘故。若通过适当的办法增强了抵御 DDoS 的能力，也就意味着加大了攻击者的攻击成本，那么绝大多数攻击者将因无法继续下去而放弃，也就相当于成功地抵御了 DDoS 攻击。

2. 前端 CDN 缓存

对于图片量较多的电子商务网站和新闻资讯类网站来说，前端 CDN 缓存的意义重大：可加快用户访问本地网站的速度，从而提升用户体验。但应该使用哪种 CDN 系统呢？这里也面临着两种选择：自行搭建 CDN 系统或租赁别人的 CDN。个人觉得自行搭建 CDN 系统是一件非常消耗财力和人力的事情，而且达不到预期目标，如果需要前端缓存，建议以租赁 CDN 为主，把更多的资金流投入到后端的文件存储和 NoSQL 缓存服务及数据库上面去。

3. 负载均衡器

负载均衡器根据它们的特点来挑选即可，LVS 的性能是最好的，特别是后端的节点超过 10 个以上时，但它对网络的要求很高，而且不支持动静分离，所以建议暂时将其作为数据库的负载均衡。HAProxy 性能优异，稳定性强，自带强大的监控页面，并且支持动静分离，我们已用 HAProxy+Keepalived 实现了亿级 PV/ 日的网站，在高并发的业务时间段，单 HAProxy 也是非常稳定，没有发生过宕机的情况。

在大公司的网站架构里，多级负载均衡也是很好的设计方案，最外面流量的负载均衡可用硬件负载均衡器（例如 F5/ NetScaler，这个是负责对流量进行转发作用的），以 Nginx 或 HAProxy 作为二层负载均衡根据频道或业务来分流。现在很多读者参考淘宝的架构，说网站最前端一定要放四层负载均衡，这个其实是针对淘宝这种巨量访问级别（十几亿 PV/日）网站的，如果是千万级 PV/日的网站，甚至是亿级 PV/日的网站，用 HAProxy/Nginx+Keepalived 基本就可以满足需求了。另外有个情况跟大家说明下，通过观察线上高流量网站的 HAProxy 负载情况，会发现 HAProxy 在高并发的情况下还是比较耗费 CPU 资源的，建议大家在此架构中采用高性能的服务器，建议使用 DELL PowerEdge R710 或更高型号的机器。另外，HAProxy/Nginx 相对于 LVS 的优势如下：

- ❑ 配置简单，语法通俗易懂。
- ❑ HAProxy/Nginx 对网络的依赖性小，理论上只要 ping 得通的网络就可以部署实施七层负载均衡。
- ❑ 根据应用配置 URI 路由规则，集中热点来提高缓存的命中率。
- ❑ 根据 URI 路由规则来进行动静分离。

4. Web 缓存层

Web 缓存层的搭建可以使用 Squid 或 Varnish。笔者在公司的不少项目中都应用过 Squid 服务器，它作为老牌的反向代理服务器，在生产环境下的稳定性是有保证的。但 Squid 对多核 CPU 支持得不好，大家可以尝试下新兴的 Varnish，它在稳定性和性能上不亚于 Squid，而且对多核 CPU 支持得也很好，性能要优于 Squid。

有的朋友可能会疑惑，为什么前端已经有了 CDN 缓存，这里还需要自己再架设一层 Web 缓存呢？如果做过高并发高流量项目的朋友应该会发现，后端 NFS 文件服务器的 I/O 压力是巨大的，有时甚至会发生拒绝服务的现象，有了这层 Web 缓存，可以起到加速后端 Web 服务及降低 NFS（或本地存储）文件服务器磁盘 I/O 压力的作用。

5. Web 服务器及 Servlet 容器

关于 Web 服务器的选择，Apache 作为 Web 的传统服务器，应用于电子商务、电子广告、页游网站都是非常稳定的，在 8GB 内存的标准配置下，抗并发能力也是非常不错的。许多公司的网站架构其实都是由最原先的一台 Apache Web 服务器发展起来的（公司高层要求平滑不中断业务升级）。如果是访问量比较大的网站，建议用 Nginx 作为 Web 服务器。

如果是每天访问量千万 PV 级别的网站，在业务高峰期间 PV 有可能过亿。推荐利用 Java 语言作为其网站核心开发语言。关于 Servlet 容器，可以考虑 Tomcat 和 Jetty，尤其是 Jetty，在我们的微信营销网站中，表现优异。我们利用 Nginx 配合 Jetty，单机能够承受两万左右的并发连接。一些 Web 聊天应用非常适合用 Jetty 做服务器，像淘宝的 Web 旺旺就是用 Jetty 作为 Servlet 容器的。Jetty 的详细工作原理及与 Tomcat 的比较可以参考文章：<https://www.ibm.com/developerworks/cn/java/j-lo-jetty/>。

6. 文件服务器层

经过后期的宣传策划，网站的客户越来越多，原先的 DRBD+Heartbeat+NFS 高可用文件服务器的磁盘 I/O 压力也越来越大，这个时候就应该考虑采用分布式文件存储方案了，MooseFS 或 GlusterFS 现在在国内也是很流行的趋势。

虽然分布式文件存储对于减轻文件服务器压力方面有所缓减，但它们占用机器的数量还是比较多的，维护起来比较复杂；而单 NFS 维护起来非常容易，事实上在有前端 CDN 和缓存层的前提下，还可以针对文件服务器进行 NFS 分组，这样从业务层面来就会更进一步减小 NFS 的压力。

现在再说下图片服务器的问题，建议大家采用独立域名而非二级域名的方式，原因如下：

- ❑ 主要是为了避免传输不必要的 Cookie，从而提升速度而且减少不必要的攻击，因为跨域是不会传输 Cookie 的。
- ❑ 多个域名可以增加浏览器并行下载的条数，因为浏览器对同一个域的域名下载条数是有限的。

7. Session 共享

Session 数据默认是在各个服务器上分别存放的，这样的话，客户端在某一次请求过后，很有可能会将请求发送到集群中的另外一台机器上，这样就会导致 Session 的丢失。所以这里采用一台独立的 Memcached 或 redis 服务器来存储整个网站的 Session 数据，然后解决各个服务器中 Session 不同步的问题。

这里不推荐将 Session 放进 MySQL 的做法，在高流量的网站中，数据库的压力是非常大的，不应该再让 Session 的问题来增加数据库方面的压力了。另外，也不推荐采用 Session 复制的方式，Session 复制的原理是通过组播的方式进行集群间的 Session 共享，比如我们常用的 Tomcat 目前就具备这样的功能。优点是 Web 容器自身支持，配置简单，这种处理 Session 的方式只适合小中型网站。缺点是当一台机器上的 Session 变更后会将变更的数据以组播的形式分发给集群间的所有节点，对网络和所有的 Web 容器都是存在开销的，集群越大浪费越严重。

运维架构师可以根据网站的实际情况来选择是否采用这种做法。

8. 数据库的压力

最后再说下数据库方面的压力，这个环节经常是整个网站的性能瓶颈所在，所以我们要在这上面投入足够多的精力。网站上线以后，如果数据库的读写压力巨大，磁盘 I/O 负载越来越高，这时候应该怎么办呢？

首先是增加数据库缓存，redis、Memcached 等 NoSQL 数据库作为数据库缓存都非常理想，他们在减轻数据库读写压力方面效果显著，事实上，像很多业务数据，放在 redis 的效果要比放在 MySQL 里面好得多，比如 IP List 业务数据，一次导入量动辄十几亿条，放在

redis 里面的读取速度要远远优于 MySQL, 同时也会大大减轻 MySQL 数据库的压力。在这里大家需要注意一个情况, 虽然我们可以用 redis 来提升网站性能, 但也有一个弊端: 如果需要 Cache 的数据对象非常多的时候, 应用程序要增加的代码量就会很多, 同时网站复杂度及维护成本也在直线上升, 这个时候开发部门和系统部门的同事们就要协同工作了。

1) 数据库架构可以采用一主多从, 读写分离的方案, 用 LVS+Keepalived 作为从数据库的负载均衡器, 通过程序实现读写分离, 前后台业务逻辑分离, 针对后台的查询我们全部转到 Slave 机器上, 这样就算查询的业务量再大的话也不会影响主要业务逻辑。

2) 对网站的业务数据库进行分库, 后面的业务是一组数据库, 如 Web、BBS、Blog 等, 对主要业务数据库进行数据的水平切分或垂直切分也是非常有必要的。

综上所述, 设计这种高流量高并发的网站系统架构, 应该尽量做到以下几点。

- ☐ 尽量把用户往外面推, 保证源站的压力小。
- ☐ 在网站测试阶段尽量做好压力测试的工作。
- ☐ 保证网站的高可用。
- ☐ 保证网站的高可扩展性。
- ☐ 多利用 NoSQL 来减轻后端数据库的压力。
- ☐ 合理优化数据库。

做到了以上几点, 我们的网站应该能够承受更大流量和并发量的冲击。

8.4 亿级 PV 高性能高并发网站架构设计

事实上, 如果我们的网站每天能达到亿级 PV 甚至 10 亿级 PV 的访问量, 那么这个数字是一个相当惊人的数字, 这么大流量的进出量, 对系统整体水平的要求都是很高的, 不仅仅是服务器层面的压力, 代码、数据库、缓存乃至文件系统都是有要求的。对于一个高并发高流量的网站来说, 任何一个环节的瓶颈都会造成网站性能的下降, 影响用户的体验, 从而造成无法弥补的损失。下面就以目前正在维护的 DSP 大型电子广告系统来举例说明, 6 个数据中心, 每天日 PV 接近 10 亿, 平均 30 多万 QPS, 业务机器单机并发连接数 2.2 万以上。

考虑到业务涉及世界各地, 6 个数据中心需要进行全球化部署, 业务高峰期间能够快速增添机器以应付暴增流量, 并且业务需要进行 Hadoop/Spark 分析数据, 还要考虑稳定的存储文件系统等, 而这些 AWS 都有相对应的产品, 可以极大地简化运维成本。因此最终考虑采用 AWS 云计算平台。

1. 负载均衡层

实际上, 网站面对这么大的流量冲击, 我们的第一反应应该就是采用一级分流的方式, 一般来说, DNS 轮询是一种常用的做法, 我们可以利用 DNS 轮询将流量第一时间分散到各个数据中心, 这里其实用到了分布式的思想, 这样做就会不至于让其中的一个数据中心因为顶不住流量而出现宕机的情况, 这里我们用的是 PowerDNS。

在这种流量规模的系统中，还应该关注另外两个参数 QPS 及系统响应时间，QPS 即 Queries Per Second，意思是“每秒查询率”，是系统每秒能够响应的查询次数，是对一个特定的查询服务器在规定时间内所处理流量多少的衡量标准。之所以应该关心这个数值，是因为它是系统整体性能的重要参考标准。

系统响应时间是我们非常关注和重视的另一个方面。由于我们的 DSP 广告主服务平台会参与 RTB (Real Time Bidding) 实时竞价，所以整个响应时间越短，效果会越好。RTB 实时竞价，是一种利用第三方技术在数以百万计的网站或移动端，针对每一个用户的展示行为进行评估及出价的竞价技术。所以从参与 RTB 实时竞价开始到完成，整个过程应尽量控制在 500 ~ 600ms 之间，如果此响应时间过大，则会失去竞争优势。

中间层的负载均衡器采取的也是常见架构的做法，用的是 Nginx。因为是 AWS EC2 机器，每台机器能够分配的带宽有限，机器很容易被流量打满。所以我们除了采用常规的开源软件监控流量以外，还自己开发了监控工具来进行监控。

我们选用 Nginx 作为其中间层的负载均衡，作用有以下几点：

- ❑ 七层负载均衡，实现各种规则转发。
- ❑ 管理业务接口。
- ❑ 灰度发布。我们可以利用 Nginx 的权重算法，将流量分散到线上的某台测试机器上，如果代码不能顺利通过，也只会影响到这一台测试机器。
- ❑ 反向代理静态页面缓存，加快用户访问速度。

进来的流量经过这样处理以后，基本上可以分流到各数据中心上面，但有一点要注意，Nginx 作为二级负载均衡的压力很大，平时要注意以下两种情况。

- ❑ 监控 Nginx 的系统负载内核有无报错及 CPU 利用率情况。
- ❑ 监控其带宽总体使用情况。
- ❑ 后端 bidder 机器的响应时间。

基于系统响应时间的考虑，核心平台后期直接删除 Nginx 负载均衡层，最前端的 DNS 轮询直接连接后端的 bidder 机器，bidder 机器的响应时间在 50ms 左右。

2. Web 应用服务器

我们的主要业务机器是 bidder 机器，用于竞标价格。bidder 机器这块选用的是 Nginx+Lua (ngx_lua 模块)，那么什么是 ngx_lua 模块呢？

ngx_lua 是 Nginx 的一个模块，将 Lua 嵌入到 Nginx 中，从而可以使用 Lua 来编写脚本，这样就可以使用 Lua 编写应用脚本，然后部署到 Nginx 中运行，即 Nginx 变成了一个 Web 容器；这样开发人员就可以使用 Lua 语言开发高性能的 Web 应用了。

ngx_lua 提供了与 Nginx 交互的很多 API，对于开发人员来说只需要学习这些 API 就可以进行功能开发了，而对于开发 Web 应用来说，如果接触过 Servlet 的话，其开发和 Servlet 类似，无外乎就是知道接收请求、参数解析、功能处理、返回响应这几步的 API 是什么样子。

理论上可以使用 ngx_lua 开发各种复杂的 Web 应用,不过 Lua 是一种脚本/动态语言,不适合业务逻辑比较重的场景,适合小巧的应用场景,代码行数保持在几十行到几千行。目前见到的一些应用场景有以下几种。

- ❑ Web 应用:会进行一些业务逻辑处理,甚至进行耗 CPU 的模板渲染,一般流程为 MySQL/redis/HTTP 获取数据→业务处理→产生 JSON/XML/模板渲染内容,比如京东的列表页或商品详情页。
- ❑ 接入网关:实现如数据校验前置、缓存前置、数据过滤、API 请求聚合、AB 测试、灰度发布、降级、监控等功能。
- ❑ Web 防火墙:可以进行 IP/URL/UserAgent/Referer 黑名单、限流等功能。
- ❑ 缓存服务器:可以对响应内容进行缓存,减少到后端的请求,从而提升性能。
- ❑ 其他:如静态资源服务器、消息推送服务、缩略图裁剪等。

线上系统主要用 AWS 的 c3.xlarge (4vCPU、14GB) 来运行 Nginx+Lua,在线上运行时,若并发连接数超过 2.4 万,则带宽基本就会被打满(虽然 AWS 没有带宽限制,但是由于多虚拟机共享了物理机的网络性能和 I/O 性能,因此导致 c3.xlarge 的带宽限制大约在 40MB ~ 50MB 之间),CPU 利用率在 90% 左右,系统负载不到 4。

为了处理业务高峰期的流量,一般会增添 10 台左右 bidder 机器,AWS 的 AMI (映像复制) 功能非常方便,而且 Instance 可以按照小时收费,这样也可极大地降低运营成本。

我们的网站前台主要是针对客户的,跟常见的 CMS 系统类似,开发语言主要是 PHP,Web 框架选用的轻量级 CI (Code Igniter)。Hadoop/Spark 数据分析这块则主要是 Java 和 Python。由于整个系统涉及的开发语言比较多,因此以 Python 作为胶水语言,把系统的各个子模块都衔接了起来。另外,运维自动化的主要开发语言也是 Python。

参考文档:

<http://www.tuicool.com/articles/VjMZF3j>

<http://jinnianshilongnian.iteye.com/blog/2258111>

3. Session 共享的问题

由于 DSP 广告系统提供的是服务 (bidder 机器发送的是竞价请求),即无状态的 HTTP 访问请求,并非传统型的 Web 网站,所以此系统并不需要关心这个问题,而大型 Web 网站肯定必须要关注 Session 共享的问题,建议大家利用 redis 缓存服务器来解决此问题,其优势就是快,快速进行大量 Session 数据的存储和读取,redis 缓存服务器完全可以胜任这份工作。而 redis 的主从方案,可以避免 redis 的单点问题。

4. 数据缓存层

因为 DSP 系统产生的大量的 ip list、domain、关键词等数据需要快速、有效地读取,之前考虑将这些数据放在 MySQL 数据库上,后面发现存在着速度问题,而且 ip list 数据量太大,每次导入都是十几亿条,这样我们就需要一个 NoSQL 的解决方案。在比对测试

了 Memcached 和 redis 的速度和效率后，最终选择了 redis。在最终将 redis 应用到线上环境时，我们也在软硬件及数据结构方面对 redis 进行了优化，主要工作有如下几个方面：

- ❑ 选用了 EC2 内存型实例 (r3.xlarge 或 r3.2xlarge) 来运行 redis 机器。
- ❑ 针对 redis 数据结构进行了重组优化。
- ❑ 将运行 redis 集群机器的数量提高到了 10 台到 15 台左右（每个数据中心的数据会不一致）。

我们是通过自己开发的一致性哈希算法程序来统一管理 redis 机器实例的，便于控制，但也有不少的问题：比如支持失败节点自动删除、程序的单点故障等。目前也在尝试使用 Twitter 的 Twemproxy 程序来管理这些 redis 机器。

Twemproxy 通过引入一个代理层，可以将其后端的多台 redis 实例进行统一管理与分配，使应用程序只需要在 Twemproxy 上进行操作，而不用关心后面具体有多少个真实的 redis 实例机器。

Twemproxy 有很多优势，我们比较关心的有以下几点。

(1) 支持失败节点自动删除。

- ❑ 可以设置重新连接该节点的时间。
- ❑ 可以设置连接多少次之后删除该节点。
- ❑ 该方式适合作为 Cache 存储。

(2) 支持设置 HashTag

通过 HashTag 可以自己设定将两个 KEY 哈希到同一个实例上去。

(3) 减少与 redis 的直接连接数

- ❑ 保持与 redis 的长连接。
- ❑ 可设置代理与后台每个 redis 连接的数目。

(4) 自动分片到后端多个 redis 实例上

- ❑ 多种哈希算法：能够使用不同的策略和哈希函数支持一致性哈希。
- ❑ 可以设置后端实例的权重。

(5) 避免单点问题

可以平行部署多个代理层，客户端会自动选择可用的那一个。

引入 Twemproxy 以后，分布式的 redis 集群中出现的许多问题都可以解决。

5. MySQL 数据库和数据仓库

由于数据读取压力全部在 redis 集群上面，分散到后端的 MySQL 上的压力就非常小了，因此对于 MySQL，我们用的是最普通的 MySQL 一主一从，主要存放系统前台的表数据，数据量并不大。不过，目前业务数据库的核心表只是单维度的，后期的业务表可能会向多维度方向转变，这样会导致 MySQL 的磁盘容量呈现一个暴增趋势，所以现在在考虑使用 Amazon Redshift 数据仓库服务。

什么是 Amazon Redshift？可参考官方资料文档：<http://aws.amazon.com/cn/redshift/>。

Amazon Redshift 是一种快速、强大且完全托管的 PB 级云中数据仓库服务。客户可以以每小时 0.25 USD 的价格从小做起，无需订立长期合约或预付费，然后以 1TB 1000 USD / 年的价格再扩展到 1PB 或以上，这个费用比大多数其他数据仓库解决方案成本的十分之一还要低。传统的数据仓库需要相当数量的时间和资源来进行管理，尤其是大型数据集。另外，内部部署型数据仓库的建立成本、维护及日益增长的自我管理相关的财务成本也非常之高。Amazon Redshift 不仅大大降低了数据仓库的成本，而且还能轻而易举地对大量数据进行快速分析。

Amazon Redshift 利用基于 SQL 的常用客户端及商业智能 (BI) 工具，通过标准的 ODBC 和 JDBC 连接，提供了对结构化数据进行快速查询的功能。查询为多个物理资源之间的分布式并行查询。在 AWS 管理控制台中点击几次或调用一个 API 即可轻松地对 Redshift 数据仓库进行扩展或缩减。Amazon Redshift 自动修补数据仓库并将其备份，并按照用户定义的保留期存储备份。Amazon Redshift 利用复制和连续备份来提高可用性并改善数据持久性，从而能从组件或节点故障中自动恢复。此外，为了保护您的中转数据和静态数据，Amazon Redshift 支持 Amazon 虚拟私有云 (Amazon VPC)、SSL、AES-256 加密和硬件安全模块 (HSM)。

Amazon Redshift 有哪些优势呢？

(1) 专为数据仓库而优化

Amazon Redshift 使用各种创新技术，对于大小在 100GB 到 1PB 或更高的数据集，拥有很强的查询能力。它使用列式存储、数据压缩及区域映射，降低了执行查询所需的 I/O 数量。Amazon Redshift 拥有大规模并行处理 (MPP) 数据仓库架构，可对 SQL 操作进行并行分布处理，以利用所有可用的资源。基础硬件均为高性能数据处理而设计，使用本地附带的存储空间以最大化处理器与驱动器之间的吞吐量，同时使用 10GigE 网状网络以最大化节点之间的吞吐量。

(2) 可扩展

仅需在 AWS 管理控制台中点击几次或通过一个简单的 API 调用，就能在性能或容量需要改变时，轻松地改变云数据仓库中的节点数或节点类型。通过密集存储 (DS) 节点，可以以非常低的价格使用硬盘 (HDD) 创建超大型数据仓库。通过密集计算 (DC) 节点，可以使用高速 CPU、大量 RAM 和固态硬盘 (SSD) 创建超高性能数据仓库。利用 Amazon Redshift，只要使用单个 160GB DC1.Large 节点即可开始，并能一路扩展到使用 16TB DS2.8XLarge 节点的 1 PB 或更多压缩用户数据。调整大小时，Amazon Redshift 可将现有的集群置于只读模式，并预配置一个你选定大小的新集群，然后将数据从旧集群并行复制到新集群。在配置新集群的同时，可继续对旧集群进行查询。一旦数据被复制到新集群，Amazon Redshift 会自动将查询重新定向至新集群，并移除旧集群。

6. Amazon S3 文件系统

我们一般是利用 Amazon EMR 来运行 Hadoop/Spark 数据，业务高峰期间还需要开启大

量的 Spot Instance 机器来并行处理数据，数据分析及访问的海量日志均存放在 Amazon S3 文件系统上面进行分析汇总再交由下端的业务系统来处理。Amazon S3 具有高度持久性、可扩展性、安全性、快速且物美价廉的存储服务。借助 EMR 文件系统，Amazon EMR 可以将 Amazon S3 安全高效地用作 Hadoop 的对象存储。Amazon EMR 对 Hadoop 进行了大量的改进，因此我们可以无缝地处理 Amazon S3 中存储的大量数据。

7. DevOps

目前通过自动化运维工具和平台组同事们的努力，已经完成的工作有如下几个方面：

- ❑ bidder 业务机器的自动增加或删减。
- ❑ 分布式爬虫程序的 Spot Instance 自动增加或删减。
- ❑ redis 的一致性哈希算法程序的逐步完善。
- ❑ 线上机器的公私钥批量自动更换或增加。
- ❑ 线上机器的代码、配置文件批量自动同步。
- ❑ 图片服务做成了 CDN 模式以便于提供对外服务，增强用户体验。
- ❑ 强大的预警系统和报警系统。

下一步 DevOps 要进行的工作：增大自动化配置管理工具 Ansible 在系统中的应用比重，跟 AWS 真正地结合起来。选择 Ansible 主要是因为拥有丰富的相关支持，包括现有的很多组件、模块、开源的 Ansible 部署和脚本。笔者也尝试了市面上所有的自动化运维和自动化配置工具，发现 Ansible 是对 AWS 支持得最好的一个。Ansible 的开发过程是写大量的 Playbook。现在 Ansible 支持的有 251 个模块，特别是对于云服务的支持。像 AWS、Docker、OpenStack，部署脚本都放在一个子目录下。这就意味着把别人写的脚本拿过来，或者把别人写定义的 Playbook 拿过来非常容易。现在关于 Ansible 的开源脚本数量庞大，大约有 3000 多个项目，相信这个数字只会越来越多，这也意味着以后的很多 DevOps 工作会越来越简单容易。

8. 压力测试及其他

系统上线前，测试组的同事会用 Load Runner 对主要业务机器 bidder 进行大量的压力测试工作。系统上线前及上线后，我们也会密切关注以下方面：

- ❑ 系统整体的 QPS 情况，尤其是在业务的高峰时期。
- ❑ 系统的整体响应时间及 bidder 机器和其他业务机器的响应时间。
- ❑ bidder 机器的并发连接总数、内核有无报错和带宽的使用情况。
- ❑ bidder 机器的负载、CPU 利用率、内存使用情况。
- ❑ redis 分布式集群机器的内存使用情况。
- ❑ Scrapy 分布式爬虫的 Spot Instance 的运行情况。
- ❑ ElasticSearch 集群的全文搜索的效率。

经过同事们的共同努力和研究，我们的业务平台已经能承受更大流量的冲击，功能方

面也在日趋完善。相信到最后，我们的业务平台一定能设计出最具性价比的方案。

8.5 细分五层解说网站架构

目前网站架构一般分为网页缓存层、负载均衡层、Web 服务器层、文件服务器层、数据缓存层及数据库层，一共五层，这样在后面的讨论过程中，就可以依次用这五层对网站架构进行讨论。为了更具说服力，下面将以笔者维护过并且正在维护的较大的生产环境来举例说明。

1. 网页缓存层

首先说网页缓存层，比如 CDN 的租赁，其效果比公司自己部署 Squid/Varnish 更好更专业，毕竟 Bind View 需要精准细分，而且价格相当低廉，所覆盖的城市也更多，故而推荐采用 CDN 租赁的方式。

很多朋友喜欢尝试自建 CDN，这是一个吃力不讨好的活儿，未必能达到预期目标，关于这块运维架构师在网站架设初期就应该规划好，不要等到网站流量及压力巨大时才去规划。事实上，这一层有很多优秀的开源软件都能胜任这块工作，比如传统的 Squid。另外，后起之秀 Nginx 和 Varnish 因为性能优异，越来越多的朋友尝试在自己的网站使用它们作为自己的网页缓存。事实上，Nginx 已经具备 Squid 所拥有的 Web 缓存加速功能。此外，Nginx 对多核 CPU 的利用胜过 Squid，现在越来越多的架构师都喜欢将 Nginx 同时作为“负载均衡服务器”与“Web 缓存服务器”来使用，大家可以根据自己网站的情况，来决定究竟使用哪种软件来对自己的网站提供反向代理加速服务。

2. 负载均衡层

我们熟悉的开源软件技术有 LVS/HAProxy，还有 Nginx，它们的性能都是非常优异的。HAProxy 可能大家不是特别熟悉，但 HAProxy 在生产环境下确实表现优异，拥有强大的吞吐能力，稳定性也能与硬件相媲美，并且淘宝也在大规模地推广使用 HAProxy，有兴趣的朋友可以关注一下。

建议负载均衡分成两级来处理，一级是流量四层分发，二级是应用层面七层转发，即业务层面。首先可以通过 LVS 或 HAProxy 将流量转发给二层负载均衡（一般为 Nginx），即实现了流量的负载均衡，此处可以使用如轮询、权重等调度算法来实现负载的转发；然后二层负载均衡会根据请求特征再将请求分发出去。此处为什么要将负载均衡分为两层呢？

1) 第一层负载均衡应该是无状态的，方便水平扩容。我们可以在这一层实现流量分组（内网和外网隔离、爬虫和非爬虫流量隔离）、内容缓存、请求头过滤、故障切换（机房发生故障后切换到其他机房）、限流、防火墙等一些通用型功能，无状态设计，可以水平扩容。

2) 二层 Nginx 负载均衡可以实现业务逻辑，或者反向代理到如 Tomcat，这一层的 Nginx 跟业务有关联，可实现业务的一些通用逻辑。这一层如果可能的话，也要尽量设计成

无状态设计，方便水平扩容。

3. Web 服务器层

Web 层压力比较大的网站现在都将 Web 主要应用服务器换成了 Nginx，事实上，它在抗并发能力和稳定性方面确实超过了预期。另外，Linux 集群有一个优势，就是它的高扩展性，特别是水平（横向）扩展。就算网站的并发连接数有 10 万以上，也无非是多加几台 Web 机器（廉价的 PC 服务器也是可行的）。在实际的线上维护时笔者发现，即使是在高峰期间，实际上每台 Web 的并发并不算是特别大，所以网站的压力在这一层也能通过技术手段加以克服。

4. 文件服务器层

现在大家的生产服务器一般是使用下面的方案：

1) 单 NFS 作为文件服务器，这样的好处是维护方便，但存在着单点故障的问题，NFS 机器出现故障时需要人为手动干预。

2) NFS 分组，虽然这样可以分摊压力，但一样也存在着单点故障的问题，出现故障时需要人为手动干预。

3) DRBD+Heartbeat+NFS 高可用文件服务器，维护方便，也不存在着单点故障的问题，但随着访问量的增大，后期一样存在着压力过大的问题。

4) 采用分布式文件系统。

文件服务器磁盘 I/O 压力过大，这也是一个常见的问题，我们在维护自己的网站时，通常采取的做法有以下几点。

- 对于静态内容，如 CSS、JS、HTML 还有图片文件，可以通过租赁 CDN 的方式来处理。

- 将图片服务器独立出来，并分配独立域名。

- 磁盘的优化：将程序的读写缓存区设置得尽可能大一些。这样做的好处是：程序不是每次调用都直接写磁盘，而是先缓存到内存中，等缓存区满了再写入磁盘。

- 在适当的场景采用分布式文件系统，例如 MooseFS。MooseFS 易用、稳定，对海量小文件尤其高效，而且新版的 MooseFS 解决了 Master Server 存在的单点故障的问题，文档和社区也非常成熟，国内越来越多的公司都在使用 MFS。事实上，分布式文件系统是解决文件服务器压力过大的最终途径。但是凡事总是有利有弊，越是功能强大的东西越是复杂。随着网站功能的增多，摊子越大，机器越多，维护起来就会越复杂，这样会极大地增加运维人员的工作难度。

大家可以尝试根据自己网站的情况，来决定究竟选择哪一种开源软件作为自己的文件服务器。

5. 数据缓存层和数据库层

网站的 PV、UV 及 QPS 和并发连接数增加以后，数据库这块的压力是最大的，数据库

的压力归根结底还是磁盘的 I/O 压力。

Oracle RAC 是很成熟的商业分布式方案，它保证了数据的高可用性，当然了价格也是非常昂贵的（如果使用了高配置的 PC 服务器。Oracle 一般按照 CPU 的个数来收费）；那么如果使用免费的开源方案，例如 MySQL 数据库，面对这种数据库磁盘 I/O 压力大的情况，应该如何处理呢？

首先在业务逻辑上将数据进行分离。很多读写频繁的业务数据，比如 ip list 和 domain 等信息都没有必要用 MySQL 数据库来保存，我们利用 redis 分布式缓存来保存这些数据，这样读取速度也能得到保证，后端 MySQL 数据库的压力也可以得到缓减。

其次，数据库的硬件方面可以考虑投入磁盘阵列做成 RAID 10，如果资金充裕，磁盘可以用 SSD 固态硬盘来代替 SAS 机械硬盘。

必须要合理地设计 MySQL 数据库的架构，事实上，在生产环境下，一主多从、读写分离是比较靠谱的设计方案，对于 MySQL 的负载均衡，这里推荐大家使用 LVS/DR，这是因为从机 MySQL 节点机器超过 10 台时，HAProxy 的性能将不如 LVS/DR。

如果网站的业务量过大，还可以采用分库的方法，比如将网站的业务量分成 Web、Blog、Mall 等几组，每一组均采用主从架构，这样设计的话就避免了单组数据库压力过大的情况。

最后，还应该配合公司的 MySQL DBA，在数据库参数优化、SQL 语句优化、数据切分上多做功夫，避免让 MySQL 数据库成为网站的瓶颈。必要的时候，考虑分布式 SQL 解决方案，例如 Redshift 及 Hbase 等。

希望大家能够通过以上对网站的五层分解，结合自己网站的情况，了解每一层在网站设计中的作用和重要性，找出网站瓶颈加以优化，将自己的网站打造成高可用高可扩展性的网站。

8.6 小结

本章以笔者维护过的百万级、千万级、亿级 PV 高可用高流量网站架构为例来说明网站的系统架构设计，并且细分为五层来解说网站的架构设计。在实际的工作中，系统架构设计绝对不是一项轻松的工作，如果大家能从这些案例中学习到对自己有帮助的技能点，提升自己的专业水平，优化自己的网站，提升用户的体验，那么笔者将甚感欣慰。

HAProxy 1.4 的配置文档

下面是 HAProxy 1.4 的一些常用配置，这些配置可实现 HAProxy 的一些常用功能。大家在写自己的 HAProxy 配置文件时，可以对比参考下此配置文档。

配置的具体实例如下所示。

```
global
```

全局的日志配置，其中日志级别是 [err warning info debug]。

日志设备必须为如下 24 种标准 syslog 设备中的一种：

```
kern、user、mail、daemon、auth、syslog、lpr、news  
uucp、cron、auth2、ftp、ntp、audit、alert、cron2  
local0、local1、local2、local3、local4、local5、local6、local7
```

这里推荐 local3，其日志设备格式如下：

```
log 127.0.0.1 local3 info #[err warning info debug]
```

下面为最大连接数：

```
maxconn 4096
```

用户（推荐用 HAProxy 用户）：

```
user nobody
```

用户组（推荐用 HAProxy 用户组）：

```
group nobody
```

使 HAProxy 进程进入后台运行，这是推荐的运行模式：

```
daemon
```

创建 4 个进程进入 daemon 模式运行。此参数要求将运行模式设置为“daemon”:

```
nbproc 4
```

将所有进程的 pid 写入文件, 启动进程的用户必须有访问此文件的权限:

```
pidfile /home/admin/haproxy/logs/haproxy.pid
defaults
```

默认的模式 mode, 有 3 个参数值可选: {tcp|http|health}, tcp 是 4 层, http 是 7 层, health 只会返回 OK, 这里大家可以根据实际情况进行选择:

```
mode http
```

采用 HTTP 日志格式:

```
option httplog
```

三次连接失败就认为是服务器不可用, 此数值可以通过修改后面的数字来设置:

```
retries 3
```

如果 Cookie 写入了 ServerID 而客户端不会刷新 Cookie, 那么待 ServerID 对应的服务器挂掉后, 将强制定向到其他健康的服务器上:

```
option redispatch
```

当服务器负载很高的时候, 自动结束掉当前队列处理比较久的链接:

```
option abortonclose
```

默认的最大连接数:

```
maxconn 4096
```

表示连接超时:

```
timeout 5000
```

表示客户端超时:

```
clitimeout 30000
```

表示服务器超时:

```
srvtimeout 30000
```

表示心跳检测超时:

```
timeout check 1000
```



注意 一些参数值为时间, 比如说 timeout。时间值通常的单位为毫秒 (ms), 但是也可以通过加 # 后缀, 来使用其他的单位。

下面是统计页面的配置:

```
listen admin_stats
```

监听端口:

```
bind 0.0.0.0:1080
```

http 的 7 层模式:

```
mode http
```

日志设置:

```
log 127.0.0.1 local0 err #[err warning info debug]
```

统计页面自动刷新时间:

```
stats refresh 30s
```

统计页面的 URI:

```
stats uri /admin_stats
```

统计页面密码框上的提示文本:

```
stats realm Gemini\ Haproxy
```

统计页面的用户名和密码设置:

```
stats auth admin:admin101
```

隐藏统计页面上 HAProxy 的版本信息:

```
stats hide-version
```

下面是网站检测的 listen 定义:

网站健康检测 URL, 用来检测 HAProxy 管理的网站是否可用, 它是依靠检查后端的 Web 服务器是否存在 index.php 来判断后端主机是否挂掉的; 如果后端的所有 Web 机器上均没有 index.php 或都挂掉了, 那么我们访问 HAProxy 主机地址时, 例如 http://192.168.1.103 时, 浏览器就会返回如下报错信息:

```
503 Service Unavailable, No server is available to handle this request
```

网站检测的 listen 格式为:

```
listen web_proxy 192.168.1.103:80
```

关于监听的其他配置选项, HAProxy 默认设置已经做好了, 建议不要太多人为的干预。

下面是 frontend 配置:

```
frontend http_80_in
```

监听端口:

```
bind 0.0.0.0:80
```

http 的 7 层模式:

```
mode http
```

应用全局的日志配置:

```
log global
```

启用 http 的 log:

```
option httplog
```

每次请求完毕后主动关闭 http 通道, HA Proxy 不支持 keep-alive 模式:

```
option httpclose
```

如果后端服务器需要获得客户端的真实 IP 则需要配置此参数, 以便从 HTTP Header 中获得客户端 IP:

```
option forwardfor
```

下面是 HAProxy 的日志记录内容配置:

```
capture request header Host len 40
capture request header Content-Length len 10
capture request header Referer len 200
capture response header Server len 40
capture response header Content-Length len 10
capture response header Cache-Control len 8
```

下面是 ACL 的策略定义。

如果请求的域名满足正则表达式则返回 true, -i 表示忽略大小写:

```
acl denali_policy hdr_reg (host) -i ^(www.gemini.taobao.net|my.gemini.taobao.
net|auction1.gemini.taobao.net)$
```

如果请求的域名满足 trade.gemini.taobao.net 则返回 true, -i 表示忽略大小写:

```
acl tm_policy hdr_dom (host) -i trade.gemini.taobao.net
```

如果在请求 url 中包含 sip_apiname=, 则此控制策略返回 true, 否则为 false:

```
acl invalid_req url_sub -i sip_apiname=
```

如果在请求 url 中存在 timetask 作为部分地址路径, 则此控制策略返回 true, 否则返回 false:

```
acl timetask_req url_dir -i timetask
```

当请求的 header 中 Content-length 等于 0 时则返回 true:

```
acl missing_cl hdr_cnt (Content-length) eq 0
```


下面是与 ACL 策略匹配的相应配置。

当请求的 Header 中 Content-length 等于 0 时则阻止请求，返回 403 错误：

```
block if missing_cl
```

block 表示阻止请求，返回 403 错误，如果不满足策略 invalid_req，或者满足策略 timetask_req，则阻止请求：

```
block if !invalid_req || timetask_req
```

当满足 denali_policy 的策略时则使用 denali_server 的 backend：

```
use_backend denali_server if denali_policy
```

当满足 tm_policy 的策略时则使用 tm_server 的 backend：

```
use_backend tm_server if tm_policy
```

reqisetbe 关键字定义，根据定义的关键字选择 backend：

```
reqisetbe ^Host:\ img dynamic
reqisetbe ^[^\ ]*\ / (img|css)/ dynamic
reqisetbe ^[^\ ]*\ /admin/stats stats
```

以上都不满足的时候使用默认 mms_server 的 backend：

```
default_backend mms_server
```

HAProxy 的错误页面设置如下所示：

```
errorfile 400 /home/admin/haproxy/errorfiles/400.http
errorfile 403 /home/admin/haproxy/errorfiles/403.http
errorfile 408 /home/admin/haproxy/errorfiles/408.http
errorfile 500 /home/admin/haproxy/errorfiles/500.http
errorfile 502 /home/admin/haproxy/errorfiles/502.http
errorfile 503 /home/admin/haproxy/errorfiles/503.http
errorfile 504 /home/admin/haproxy/errorfiles/504.http
```

下面是 backend 的设置：

```
backend mms_server
```

http 的 7 层模式：

```
mode http
```

负载均衡的方式，roundrobin 平均方式：

```
balance roundrobin
```

允许插入 SERVERID 到 Cookie 中，SERVERID 在后面可以如下定义：

```
cookie SERVERID
```

心跳检测的 URL，HTTP/1.1 和 Host:XXXX 指定了心跳检测 HTTP 的版本，XXX 为检

测时请求服务器的 request 中的域名, 如下所示:

```
option httpchk GET /member/login.jhtml HTTP/1.1\r\nHost:member1.gemini.taobao.net
```

服务器定义, cookie 1 表示 SERVERID 为 1, check inter 1500 是检测心跳频率。rise 3 是 3 次正确则认为服务器可用, fall 3 是 3 次失败则认为服务器不可用, weight 代表权重:

```
server mms1 10.1.5.134:80 cookie 1 check inter 1500 rise 3 fall 3 weight 1
server mms2 10.1.6.118:80 cookie 2 check inter 1500 rise 3 fall 3 weight 2
backend denali_server
mode http
```

负载均衡的方式, source 是根据客户端 IP 进行哈希的方式:

```
balance source
```

如果设置了 backup, 会默认第一个 backup 优先, 设置 option allbackups 后所有备份服务器的权重都是一样的:

```
option allbackups
```

心跳检测 URL 设置:

```
option httpchk GET /mytaobao/home/my_taobao.jhtml HTTP/1.1\r\nHost:my.gemini.taobao.net
```

可以根据机器性能的不同, 不使用默认的连接数配置而使用自己的特殊连接数配置, 比如 minconn 10 maxconn 20:

```
server denlai1 10.1.5.114:80 minconn 4 maxconn 12 check inter 1500 rise 3 fall 3
server denlai2 10.1.6.104:80 minconn 10 maxconn 20 check inter 1500 rise 3 fall 3
```

备份机器的配置, 正常情况下不会使用备机, 当主机全部服务器都宕机的时候备份机才会启用:

```
server dnali-back1 10.1.7.114:80 check backup inter 1500 rise 3 fall 3
server dnali-back2 10.1.7.114:80 check backup inter 1500 rise 3 fall 3
backend tm_server
mode http
```

负载均衡的方式, leastconn 根据服务器当前的请求数, 取当前请求数最少的服务器:

```
balance leastconn
option httpchk GET /trade/itemlist/prepayCard.htm HTTP/1.1\r\nHost:trade.gemini.taobao.net
server tm1 10.1.5.115:80 check inter 1500 rise 3 fall 3
server tm2 10.1.6.105:80 check inter 1500 rise 3 fall 3
```

下面是 reqisetbe 自定义的关键字匹配 backend 的部分:

```
backend dynamic
mode http
balance source
option httpchk GET /welcome.html HTTP/1.1\r\nHost:www.taobao.net
```

```

server denlai1 10.3.5.114:80 check inter 1500 rise 3 fall 3
server denlai2 10.4.6.104:80 check inter 1500 rise 3 fall 3
backend stats
mode http
balance source
option httpchk GET /welcome.html HTTP/1.1\r\n Host:www.163.com
server denlai1 10.5.5.114:80 check inter 1500 rise 3 fall 3
server denlai2 10.6.6.104:80 check inter 1500 rise 3 fall 3

```

参考文档:

<http://haproxy.org/download/1.4/doc/configuration.txt>

<http://haproxy.org/>

rsync 及 inotify 在工作中的应用

大家应该很熟悉和了解 Linux 下的 rsync 工具了吧，rsync (remote synchronize) 是一个远程数据同步工具，可通过 LAN/WAN 快速同步多台主机间的文件。rsync 使用 rsync 算法来使本地主机和远程主机之间的文件达到同步，这个算法并不是每次都整份传送，它只传送两个文件的不同部分，因此速度相当快。

rsync 的优点如下：

- ☐ 可以镜像保存整个目录树和文件系统。
- ☐ 可以很容易做到保持原来文件的权限、时间、软硬链接等。
- ☐ 无需特殊权限即可安装。
- ☐ 拥有优化的流程，文件传输效率高。
- ☐ 可以使用 RSH、SSH 等方式来传输文件，当然也可以直接通过 Socket 连接。
- ☐ 支持匿名传输。

另外，与 scp 相比，它们的传输速度不是一个数量级的。我们在局域网时经常用 rsync 和 scp 传输大量数据文件，发现 rsync 在速度上至少比 scp 快 20 倍以上，这得益于 rsync 强大的 checksum 算法。所以大家如果需要在 Linux 服务器之间传输大数据时，rsync 是最好的选择。rsync 2.6.8 版本中存在 Bug，不过在 2.6.9 版本中已解决。另外，rsync 3.0 版本的算法相对于 rsync 2.x 的算法也有所改善，3.0 是边对比边同步，2.x 是完全对比之后再同步，3.0 的效率肯定更高，所以这里推荐大家采用 3.0 或更高级的版本。CentOS 6.4 x86_64 下我们可以用如下命令来查看 rsync 的版本号，如下所示：

```
rsync --version
```

命令显示结果如下所示：

```
rsync version 3.0.6 protocol version 30
Copyright (C) 1996-2009 by Andrew Tridgell, Wayne Davison, and others.
Web site: http://rsync.samba.org/
Capabilities:
  64-bit files, 64-bit inums, 64-bit timestamps, 64-bit long ints,
  socketpairs, hardlinks, symlinks, IPv6, batchfiles, inplace,
  append, ACLs, xattrs, iconv, symtimes
rsync comes with ABSOLUTELY NO WARRANTY. This is free software, and you
are welcome to redistribute it under certain conditions. See the GNU
General Public Licence for details.
```

关心 rsync 算法的朋友可以关注下面的文章，如下所示：

<http://coolshell.cn/articles/7425.html>

http://en.wikipedia.org/wiki/Rolling_hash

<http://en.wikipedia.org/wiki/Adler-32>

<http://wangyuanzju.blog.163.com/blog/static/130292010101252632998/>

<http://blog.csdn.net/liuben/article/details/5793706>

<http://blog.csdn.net/liuben/article/details/5693974>

如果遇到网页打不开的情况，推荐大家使用 shadowsocks 翻墙，下载地址为：<https://github.com/shadowsocks/shadowsocks-windows>。

B.1 rsync 的应用模式

笔者建议将安装及提供 rsync 服务的机器称为 rsync 服务器端，没有提供 rsync 的机器称为 rsync 客户端，这样可以更好地理解本节的内容。这里笔者按照 rsync 平时在工作中的应用，将其分成了 4 种应用模式，如下所示：

1) 本地 Shell 模式，顾名思义，此操作主要是针对本地机器的，用于在本地机器上复制目录内容。

2) 远程 Shell 模式

使用一个远程 Shell 程序来实现将 rsync 服务器端的内容拷贝到本地机器，或者将本地机器的内容拷贝到 rsync 服务器端的机器中。

3) 列表模式，可以通过 rsync 查看远程机器的目录信息。

如：`rsync -v 192.168.1.204:/data/html/www/images/`

4) 服务器模式。

这个在工作中应该是应用得最多的，rsync 服务器开启 rsync 服务，用于接收 rsync 客户端的文件传输请求。

那么，究竟应该如何在 CentOS 6.4 下实现 rsync 服务呢？这里以工作机器举例说明。

具体安装步骤如下：

首先准备一台系统为 CentOS 6.4 x86_64、IP 为 192.168.1.207 的机器，将其作为 rsync

服务器，另外准备一台系统为 CentOS 6.4 x86_64、IP 为 192.168.1.204 的机器作为 rsync 客户端。

具体的安装步骤不再多说，这里只介绍重点内容。首先检查 rsync 是否安装，命令如下：

```
rpm -q rsync
```

命令显示结果如下：

```
rsync-3.0.6-12.el6.x86_64
```

如果没有安装 rsync，大家可以使用如下命令进行安装：

```
yum -y install rsync
```

另外，关闭防火墙和 SELinux，因为是在内网中传输，所以没必要打开。关闭它们，免得引起不必要的麻烦，命令如下：

```
service iptables stop
chkconfig iptables off
setenforce 0
```

下面分享一下我自己定义的配置文件 /etc/rsyncd.conf（说明：此文件并不是系统创建的，大家也可以给它取不同的名字）。先给出具体代码，后面再进行详细解释，代码如下所示：

```
uid = www
gid = www
user chroot = no
max connections = 200
timeout = 600
pid file = /var/run/rsyncd.pid
lock file = /var/run/rsyncd.lock
log file = /var/log/rsyncd.log
[www_rsync]
path=/data/html/www/images/
ignore errors
read only = no
list = no
hosts allow = 192.168.21.0/255.255.255.0
auth users = test
secrets file = /etc/rsyncd.password
```

下面说明一下 /etc/rsyncd.conf 的语法。

```
uid = www
```

上面指的是运行 rsync 的用户为 www。

```
gid = www
```

表示运行 rsync 的组为 www。

因为我们的线上系统运行 Nginx 也是用的 www:www 用户，这里为了保证 rsync 以后

文件及文件夹的权限一致，所以这里也选用了 `www:www` 用户。

```
use chroot = no
```

如果“`use chroot`”指定为 `true`，那么 `rsync` 在传输文件以前首先 `chroot` 到 `path` 参数所指定的目录下。这样做可实现额外的安全防护功能，但缺点是需要给用户以 `root` 的权限，并且不能备份通过外部的符号连接所指向的目录文件。在默认情况下，`chroot` 的值为 `true`。但是这一般是没有必要的，笔者选择 `no` 或 `false`。

```
list = no
```

表示不允许列清单。

```
max connections = 200
```

表示最大连接数。

```
timeout = 600
```

表示覆盖客户指定的 IP 超时时间，也就是说 `rsync` 服务器不会永远等待一个崩溃的客户端。

```
pidfile = /var/run/rsyncd.pid
```

指的是 `pid` 文件的存放位置。

```
lock file = /var/run/rsync.lock
```

指的是锁文件的存放位置。

```
log file = /var/log/rsyncd.log
```

指的是日志文件的存放位置。

```
[www_rsync]
```

这是认证模块名，即跟 `samba` 软件的语法是一样，是对外公布的名字。

```
path = /data/html/www/images
```

这是参与同步的目录。

```
ignore errors
```

表示可以忽略一些无关的 I/O 错误。

```
read only = no
```

表示允许读和写。

```
list = no
```

表示不允许列清单。

```
hosts allow = 192.168.1.0/255.255.255.0
```

这里跟 samba 的语法是一样的，只允许 192.168.1.0/24 网段的机器进行同步，拒绝其他一切网段连接。

```
auth users = www
```

指的是认证的用户名，这里为了权限的一致性，建议也选择 www。

```
secrets file = /etc/rsyncd.password
```

指的是密码文件的存放地址。

启动服务器端的 rsync，可通过 xinetd 来控制，这里要对 rsync 进行修改，我们先编辑 rsync 相关的文件 /etc/xinetd.d/rsync，如下所示：

```
service rsync
```

```
{
    disable = yes
    socket_type      = stream
    wait            = no
    user            = root
    server          = /usr/bin/rsync
    server_args     = --daemon
    log_on_failure  += USERID
}
```

将 disable=yes 改为 disable=no，然后重启 xinetd 即可，命令如下：

```
/etc/init.d/xinetd restart
```

配置中应该注意的问题如下所示。

- ❑ [www_rsync]：认证模块名，这个认证模块名是服务器对外的名字，机器同步时只会认这个名字。
- ❑ path = /data/html/www/images/：参与同步的目录的权限。如果此目录的权限不够，rsync 同步是成功不了的。这里建议用如下命令进行检查。

```
ls -ld /data/html/www/images/
```

确保 www:www 用户对此目录有读、写和执行权限，命令如下所示：

```
chown -R www:www /data/html/www/images/
```

- ❑ 注意用户名和密码的问题。

```
echo "www:www" >/etc/rsyncd.password
```

说明：这里我设置的是用户名和密码一致。为了安全起见，设置他的权限为 600，如下所示：

```
chmod 600 /etc/rsyncd.password
```

rsync 客户端配置：

```
echo "www" > /etc/rsyncd.password
```

这里只需要密码，不需要用户，免得要同步时还要进行手动互动，为了安全，一样配置 600 的权限，命令如下所示：

```
chmod 600 /etc/rsyncd.password
```

下面首先来说说在工作中经常遇到的 rsync 问题。

故障一：服务器端的目录不存在或无权限，故障描述如下：

```
@ERROR: chroot failed
rsync error: error starting client-server protocol (code 5) at main.c(1522)
[receiver=3.0.3]
```

解决方法：创建目录或修改目录权限。

故障二：服务器端该模块（tee）需要验证用户名和密码，但客户端没有提供正确的用户名和密码，认证失败，故障描述如下：

```
@ERROR: auth failed on module tee
rsync error: error starting client-server protocol (code 5) at main.c(1522)
[receiver=3.0.3]
```

解决方法：提供正解的用户名和密码。

故障三：服务器上不存在指定的模块，故障描述如下：

```
@ERROR: Unknown module 'tee_nonexists'
rsync error: error starting client-server protocol (code 5) at main.c(1522)
[receiver=3.0.3]
```

解决方法：提供正确的模块名。

接着我们可以进行测试工作了。

在 rsync 客户端的机器上执行如下命令：

```
rsync -vzrtopg --delete /data/html/www/images/
www@192.168.1.207::www_sync --password-file=/etc/rsyncd.password
```

这时候就可以看到正确的同步效果了，结果如下所示：

```
sending incremental file list
deleting key/xen-cms-private
deleting key/xen-cms
deleting key/id_rsa_yhc.pub
deleting key/
deleting 201601211010099/upload/1/201601/index.html
deleting 201601211010099/upload/1/201601/Thumbs.db
deleting 201601211010099/upload/1/201601/11163217052267j3.doc
deleting 201601211010099/upload/1/201601/11115238075060e0.png
deleting 201601211010099/upload/1/201601/0503525603970obn.docx
deleting 201601211010099/upload/1/201601/0503412101818iro.docx
deleting 201601211010099/upload/1/201601/
```

```

deleting 201601211010099/upload/1/201512/index.html
deleting 201601211010099/upload/1/201512/
deleting 201601211010099/upload/1/
deleting 201601211010099/upload/
deleting 201601211010099/notify/
deleting 201601211010099/
sent 108 bytes  received 9 bytes  21.27 bytes/sec
total size is 0  speedup is 0.00

```



注意 在 rsync 客户端机器上, /data/html/www/images/ 和 /data/html/www/images 进行 rsync 传输的效果截然不同。如果是 /data/html/www/images, 则会将 images 目录复制到 rsync 服务器端的 /data/html/www/images/ 目录下; 如果是 /data/html/www/images/, 则不传输目录本身, 只传输目录中的文件内容。请大家在工作中注意这点。

下面再说说工作中经常用到的 rsync 参数, 如下所示。

- ❑ -v --verbose: 详细模式输出。
- ❑ -r --recursive: 对子目录以递归模式处理。
- ❑ -p --perms: 保持文件权限。
- ❑ -o --owner: 保持文件属主信息。
- ❑ -g --group: 保持文件属组信息。
- ❑ -t --times: 保持文件时间信息。
- ❑ --delete: 表示客户端上的数据要与服务器端完全一致, 我们还是上面的例子来说明, 请看下面的 rsync 同步命令:

```

rsync -vzrtopg --delete /data/html/www/images/
www@192.168.1.207::www_sync --password-file=/etc/rsyncd.password

```

我们在这里引入了 --delete 参数, 则会使得 rsync 客户端机器的 /data/html/www/images 目录跟 rsync 服务器端的 /data/html/www/images 目录保持完全一致, 如果客户端机器上存在着 rsync 服务器端不存在的文件或目录, 则会删除。

- ❑ --delete-excluded: 删除接收端那些被该选项指定排除的文件。
- ❑ -z --compress: 对备份的文件在传输时进行压缩处理。
- ❑ --exclude= 文件或文件夹名: 指定不需要传输的文件或文件夹名。
- ❑ --include= 文件或文件夹名: 指定需要传输的文件或文件夹名。
- ❑ --exclude-from=FILE: 排除 FILE 中指定模式的文件。
- ❑ --include-from=FILE: 不排除 FILE 中指定模式匹配的文件。

另外, 我们在工作中经常遇到的一个问题是, 经常需要快速删除海量文件, 这个应该如何来实现呢?

有时候需要快速清空包含几百万个小文件的文件夹, 我们一般会采用 `rm -rf *` 的方式来处理, 但现在的服务器一般都是机械硬盘, 这样不仅速度慢, 而且磁盘 I/O 的压力也非常

大，机器负载很容易就上去了，其实这个时候我们可以用 rsync 来快速清理。

比如说我们要清理 /data/html/www/mall/Runtime 目录里的文件，应该如何操作呢？步骤比较简单，如下所示：

1) 建立一个空的文件夹，命令如下所示：

```
mkdir /tmp/test
```

2) 用 rsync 删除 /data/www/html/mall/Runtime 目录，命令如下所示：

```
rsync --delete-before -a -v --progress --stats /tmp/test/ /data/html/www/mall/Runtime
```

这样我们要删除的 /data/www/html/mall/Runtime 目录就会被清空了，删除的速度也会非常快。

选项说明分别如下。

❑ `-delete-before`：接收者在传输之前进行删除操作。

❑ `-progress`：在传输时显示传输过程。

❑ `-a`：归档模式，表示以递归的方式传输文件，并保持所有文件的属性。

❑ `-v`：详细输出模式。

❑ `-stats`：给出某些文件的传输状态。

B.2 rsync + inotify 实现数据的实时同步更新

1. rsync 的优点与不足

rsync 在 Linux 下是一个比较重要和实用的服务，从前面的内容大家应该已经了解 rsync 具有安全性高、备份迅速、支持增量备份等的优点，通过 rsync 可以解决对实时性要求不高的数据备份需求，例如定期地备份文件服务器数据到远程服务器，对本地磁盘定期做数据镜像等。

随着应用系统规模的不断扩大，对数据的安全性和可靠性也提出了更高的要求，rsync 在高端业务系统中也逐渐暴露出了很多的不足之处，首先，rsync 同步数据时，需要扫描所有文件后进行比对，然后进行增量传输。如果文件数量达到了百万甚至千万量级，那么扫描所有文件将是非常耗时的。而且正在发生变化的往往只是其中很少的一部分，这是非常低效的方式。其次，rsync 不能实时地去监测、同步数据，虽然它可以通过 Linux 守护进程的方式触发同步，但是两次触发动作一定会有时间差，这样就导致了服务端和客户端数据可能出现不一致的情况，无法在应用故障时完全恢复数据。基于以上原因，考虑采用 rsync+inotify，就可以解决这些问题了。

2. 初识 inotify

inotify 是一种强大的、细粒度的、异步的文件系统事件监控机制，Linux 内核从 2.6.13 起，加入了对 inotify 的支持，通过 inotify 可以监控文件系统上的添加、删除、修改、移动等各种细微事件，利用这个内核接口，第三方软件就可以监控文件系统中文件的各种变化

情况了，而 inotify-tools 就是这样一个第三方软件。

在上面的章节中我们讲到，rsync 可以实现触发式的文件同步，但是通过 Crontab 守护进程的方式触发，同步的数据和实际数据会有差异，而 inotify 可以监控文件系统的各种变化，当文件有任何变动时，就触发 rsync 同步，这就刚好解决了同步数据的实时性问题。

3. 安装 inotify 工具 inotify-tools

由于 inotify 的特性需要 Linux 内核的支持，在安装 inotify-tools 前要先确认 Linux 系统内核是否达到了 2.6.13 以上，如果 Linux 内核低于 2.6.13 版本，就需要重新编译内核加入对 inotify 的支持，我们的 CentOS 6.4 系统不需要担心此问题。

```
uname -r
```

命令显示结果如下所示：

```
2.6.32-573.12.1.el6.x86_64
```

然后通过 ls 来查看是否存在 /proc/sys/fs/inotify 目录，如下所示：

```
ls -lsart /proc/sys/fs/inotify/
```

此命令显示结果如下所示：

```
total 0
0 dr-xr-xr-x 0 root root 0 Feb 25 02:15 ..
0 dr-xr-xr-x 0 root root 0 Mar  2 03:36 .
0 -rw-r--r-- 1 root root 0 Mar  2 03:36 max_user_watches
0 -rw-r--r-- 1 root root 0 Mar  2 03:36 max_user_instances
0 -rw-r--r-- 1 root root 0 Mar  2 03:36 max_queued_events
```

通过以上显示结果我们应该清楚，CentOS 6.4 x86_64 是支持 inotify 的。

4. inotify 可以监控的文件系统事件

inotify 是文件系统事件监控机制，是 dnotify 的有效替代品（dnotify 是较早的内核支持的文件监控机制）。inotify 是一种强大的、细粒度的、异步的机制，它满足各种各样的文件监控需要，而不仅仅限于安全和性能。

inotify 可以监视的文件系统事件包括如下几个方面。

- ❑ IN_ACCESS：文件被访问。
- ❑ IN_MODIFY：文件被 write。
- ❑ IN_ATTRIB：文件属性被修改，如 chmod、chown、touch 等。
- ❑ IN_CLOSE_WRITE：可写文件被 close。
- ❑ IN_CLOSE_NOWRITE：不可写文件被 close。
- ❑ IN_OPEN：文件被 open。
- ❑ IN_MOVED_FROM：文件被移走，如 mv。
- ❑ IN_MOVED_TO：文件被移来，如 mv、cp。
- ❑ IN_CREATE：创建新文件。

- ❑ IN_DELETE: 文件被删除, 如 rm。
- ❑ IN_DELETE_SELF: 自删除, 即一个可执行文件在执行时删除自己。
- ❑ IN_MOVE_SELF: 自移动, 即一个可执行文件在执行时移动自己。
- ❑ IN_UNMOUNT: 宿主文件系统被 unmount。
- ❑ IN_CLOSE: 文件被关闭, 等同于 (IN_CLOSE_WRITE | IN_CLOSE_NOWRITE)。
- ❑ IN_MOVE: 文件被移动, 等同于 (IN_MOVED_FROM | IN_MOVED_TO)。



注意 上面所说的文件也包括目录。

5. rsync + inotify 企业应用案例

笔者之前的公司的 Web 应用服务器采用的是集群方案, 6 台 Nginx 之间同步代码的方案正是 rsync+inotify, 这里以 3 台机器来说明下, 即 1 台 rsync 服务器, 2 台 rsync 客户端机器, 此环境跟上面的环境区别较大, 大家不要弄混淆了, IP 分配和用途如下所示:

```
Web-Server :192.168.1.207 rsync 客户端
Web1-Client:192.168.1.204 rsync 服务器端
Web2-Client:192.168.1.205 rsync 服务器端
```

大家注意下这里的权限分配, 不要弄混淆了, Web-Server 是作为内容发布的机器, 即代码改动是在这台机器上面操作的。这里是作为 rsync 客户端, 并非 rsync 服务器端。所有机器需要同步的目录均为 /data/htdocs/www/images, 自动同步顺序均为 Web 客户端机器向 Web-Server 端机器同步, 我们这里将 Web1-Client 和 Web2-Client 配置成 rsync 的服务器端即可, 即 Web-Server 仅仅只作为 rsync 客户端。

1) inotify-tools 是用来监控文件系统的工具, 必须安装在 Web-Server (即 rsync 客户端) 机器上, 用来监控其文件系统的变化, Web-Client 机器不需要安装。

首先开始安装 inotify-tools, 我们的机器由于提前安装了 epel 源, 这里只需要通过 yum 命令安装即可, 命令比较简单:

```
yum -y install inotify-tools
```

2) Web1-Client 和 Web2-Client 机器的 rsync 服务配置比较容易, 大家可以参考上面的内容, 下面配置好 /etc/rsyncd.conf 文件, 命令如下所示:

此命令显示内容如下所示:

```
uid = www
gid = www
user chroot = no
max connections = 200
timeout = 600
pid file = /var/run/rsyncd.pid
lock file = /var/run/rsyncd.lock
```

```
log file = /var/log/rsyncd.log

[web1_sync]
path=/data/html/www/images
#web2机器此处配置为[web2_sync]
ignore errors
read only = no
list = no
hosts allow = 192.168.1.0/255.255.255.0
auth users = www
secrets file = /etc/rsyncd.password
```

然后重启 xinetd 即可，命令如下所示：

```
/etc/init.d/xinetd restart
```

记得两台 Web 机器都要配置 /etc/rsyncd.passwd 文件，rsync 的配置过程和原理请大家参考附录前面的内容，这里就不详细说明了，注意 /etc/rsyncd.password 的文件权限和内容。

3) 配置好 Web-Server 的 inotify 脚本以后，即可让其开机启动，脚本 /root/rsync-inotify.sh 内容如下：

```
#!/bin/bash
src=/data/html/www/images/
des_ip1=192.168.1.204
des_ip2=192.168.1.205

/usr/local/bin/inotifywait -mrq --timefmt '%d/%m/%y %H:%M' --format '%T %w%f' -e
modify,delete,create,attrib $src | while read file
do
    rsync -vzrtopg --delete --progress $src www@$des_ip1::web1_sync --password-
file=/etc/rsyncd.password
    rsync -vzrtopg --delete --progress $src www@$des_ip2::web2_sync --password-
file=/etc/rsyncd.password
    echo "File Synchronization is Complete!"
done
```

脚本相关解释如下。

❑ --timefmt: 指定时间的输出格式。

❑ --format: 指定变化文件的详细信息。

这个脚本的作用就是通过 inotify 监控文件目录的变化，进而触发 rsync 进行同步操作，由于这个过程是一种主动触发的操作，是通过系统内核来完成的，所以，比起那些遍历整个目录的扫描方式来，效率要高很多。然后我们将此脚本放在 /etc/rc.local 中，即在最后一行添加相关内容，/etc/rc.local 文件改动后的内容如下所示：

```
/root/rsync-inotify.sh &
```

4) 验证就很容易了，在 Web-Server 的机器的 /data/html/www/images/ 目录下新建文

件，更改文件内容，可以很欣慰地发现，两台 Web-Client 的机器对应的目录马上也会发生相应的改变，感觉非常快捷方便。

总体说来，rsync + inotify 比较适用于没有存储环境的小文件的即时同步更新的工作场景，适合中小型规模的网站。如果 Web 集群超过 10 台的话，还是应该考虑自动化配置的方式。

用 Supervisor 批量管理进程

Supervisor，简单来说，就是一个 Python 编写的进程管理器。虽然在 Shell 下面我们可以用 `nohup` 的方式将程序放在后台执行，一个或几个还比较方便，如果有很多重要的进程需要管理的话，就不方便了。我们的线上机器，一般都有大量重要的 Python 或 Shell 程序在后台运行，大家可以试想一下十几或二十几个重要的 Python 程序在后台里运行的场景，这个时候采用 Supervisor 来进行批量管理就非常方便了。工作中很常见的一个问题是：比如很不幸地，服务器出现崩溃问题或人为重启，导致所有应用程序都退出了，此时可以用 Supervisor 同时启动所有程序而不是一个一个地敲命令来启动。

Supervisor 在 CentOS 6.4 x86_64 下的安装

因为笔者的机器提前安装了 `epel` 源，所以安装就非常方便了，命令如下所示：

```
yum -y install supervisor
```

启动 Supervisor 的命令也很简单，如下所示：

```
supervisord -c /etc/supervisord.conf
```

Supervisor 的配置文件为 `/etc/supervisord.conf`，比较简单，详细配置文档可以参考其官方文档。这里不再进行详细说明，官方文档地址如下所示：<http://supervisord.org/configuration.html>

我们可以根据需要修改里面的配置。在这里，每个不同的项目，使用了一个单独的配置文件，放置在 `/etc/supervisor/` 下面，于是修改 `/etc/supervisord.conf`，加上如下内容：

```
[include]
files = /etc/supervisor/*.conf
```

这样的好处就是如果有很多进程需要管理，可以进行批量管理，或者直接在 `/etc/supervisor.conf` 文件里添加多个进程管理，这也是可以的。下面以工作中的机器来举例说明下，`/etc/supervisord.conf` 文件内容如下所示：

我们用 `cat` 命令查看 `supervisord.conf` 文件，命令如下所示：

```
cat /etc/supervisord.conf | grep -v "^#"
```

命令显示结果如下所示：

```
[unix_http_server]
file=/tmp/supervisor.sock
;UNIX socket文件路径，这里采用Socket的方式管理而非端口，前者较之后者更为安全

[supervisord]
logfile=/tmp/supervisord.log ;Supervisord日志文件路径
logfile_maxbytes=50MB ;Supervisord日志文件大小，超出会rotate
logfile_backups=10 ;日志文件保留备份数量，默认值为10
loglevel=info ;Supervisord日志级别，这里为info
pidfile=/tmp/supervisord.pid ;Supervisord的pid文件路径
nodaemon=false ;Supervisord以daemon 的方式运行
minfds=1024 ;可以打开的文件描述符的最小值，这里为1024
minprocs=200 ;可以打开的进程数的最小值，这里为200

[rpcinterface:supervisor]
supervisor.rpcinterface_factory = supervisor.rpcinterface:make_main_rpcinterface
;此处配置成默认值

[supervisorctl]
serverurl=unix:///tmp/supervisor.sock
;通过Unix Socket连接Supervisord

[program:index]
command=/usr/bin/python index.py ;程序的启动命令
directory=/home/yhc/ContentEngine/api ;程序的启动目录
autostart=true ;在Supervisord启动的时候也自动启动
autorestart=true ;程序异常退出时自动重启
user=yhc ;程序用哪个用户启动
redirect_stderr=true ;把stderr重定向到stdout，即重定向错误日志
stdout_logfile=/data/log/service_index.log ;sdout日志输出路径

[program:masterManager]
command=/usr/bin/python masterManager.py
directory=/home/yhc/ContentEngine
autostart=true
autorestart=true
user=yhc
redirect_stderr=true
stdout_logfile=/data/log/service_master.log
```

Supervisord 配置文件至少需要一个 `[program:x]` 部分的配置，来告诉 Supervisord 需要管理哪个进程。`[program:x]` 语法中的 `x` 表示进程名，会在客户端（`supervisorctl` 界面）显示，

在 `supervisorctl` 中通过这个值来对程序进行 `start`、`restart`、`stop` 等操作。

上面这个命令会进入 `supervisorctl` 的 Shell 界面，然后可以执行不同的命令了：

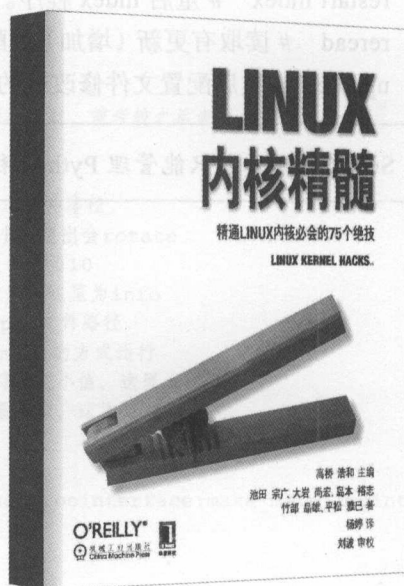
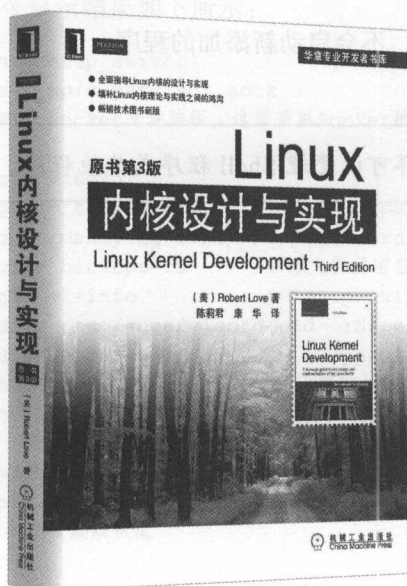
- ❑ `status` # 查看程序状态。
- ❑ `stop index` # 关闭 `index` 程序。
- ❑ `start index` # 启动 `index` 程序。
- ❑ `restart index` # 重启 `index` 程序。
- ❑ `reread` # 读取有更新（增加）的配置文件，不会启动新添加的程序。
- ❑ `update` # 重启配置文件修改过的程序。



注意

Supervisor 并非只能管理 Python 程序，同样可以管理 Shell 程序或其他程序。

推荐阅读



Linux内核设计与实现（原书第3版）

世界范围内公认的Linux内核经典著作，畅销全球多个国家

Linux内核精髓

畅销书，一线内核技术专家经验和智慧结晶，深刻解读Linux内核的资源管理、文件系统、网络、虚拟化、省电技术、调试、性能调优、分析与追踪等核心主题

推荐阅读



云计算：概念、技术与架构

作者：Thomas Erl 等 ISBN：978-7-111-46134-0 定价：69.00元



企业应用架构模式

作者：Martin Fowler ISBN：978-7-111-30393-0 定价：59.00元



设计模式：可复用面向对象软件的基础

作者：Erich Gamma 等 ISBN：7-111-07575-2 定价：35.00元



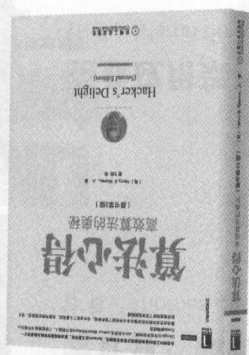
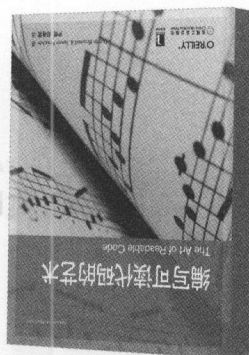
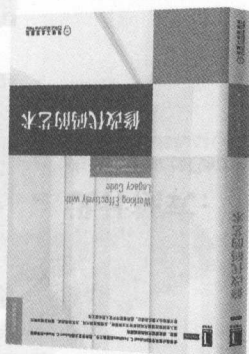
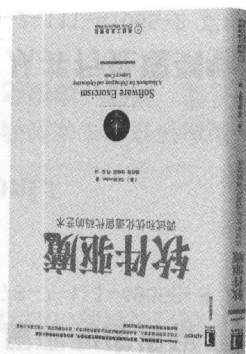
深入理解云计算：基本原理和应用程序编程技术

作者：拉库马·布亚 等 ISBN：978-7-111-49658-8 定价：69.00元



云计算与分布式系统：从并行处理到物联网

作者：Kai Hwang 等 ISBN：978-7-111-41065-2 定价：85.00元



推荐阅读

作者简介

余洪春（抚琴煮酒） 高级运维架构师、资深运维工程师，在电子商务领域及云计算领域工作10多年，在Linux集群、自动化运维、DevOps及高并发高流量网站架构设计等方面进行了深入的研究；在大量一线实践中积累了丰富的经验。精通负载均衡高可用和Python自动化运维技术，擅长高流量高性能网站架构设计。51CTO和ChinaUnix等知名社区特邀专家，ChinaUnix论坛“集群和高可用”及“监控及自动化运维技术”版版主，在社区内发表了大量技术文章，深受社区网友好评。

作者联系方式

作者邮件：

yuhongchun027@gmail.com

作者博客：

<http://yuhongchun.blog.51cto.com>

作者GitHub：

<https://github.com/yuhongchun>

随着云计算平台的流行和普及，用户的规模和访问量越来越多。一般传统型运维的PV、并发及QPS的量级都是比较有限的，重要业务都可以采用集群的方式处理，通常维护好集群就可以了。但是云计算平台所涉及的多数是服务型业务，而且机器的数量也会随着业务量的增大而增多，这个时候，自动化运维的作用就体现出来了，它能在很大程度上提高运维效率，减轻大家平时的工作量。本书旨在帮助大家掌握自动化运维的精髓，并且能熟练运用Linux集群技术设计真正适合自己公司或业务系统的网站系统架构，提升自己的职业技能。

本书的主要内容和特色：

- 书中所有Python和Shell脚本都是根据实际工作需求和业务需求而来，大家可以直接去作者的GitHub下载，然后根据自己的工作需要选择使用。
- 所有的Linux集群案例均源自工作实践和项目总结，在保证实际操作的前提下，尽可能详细解释了负载均衡高可用的理论体系，做到真正意义上的理论与实际相结合。
- 书中穿插了大量关于AWS云计算的知识点。有海外业务需求、弹性业务扩展需求或对AWS云计算感兴趣的读者可以多关注下这方面的内容。
- Python自动化运维是本书的亮点之一，笔者目前也将Fabric及Ansible用于自己公司的AWS云计算业务平台，分享出来的实际案例均源于工作中的需求，希望大家能够熟悉掌握，提升自己的职业技能。
- 对网站架构设计感兴趣的读者可以重点关注最后一章的内容，尤其是高性能高并发量的网站架构设计，这章涵盖的知识点和运维技能也比较多，仔细阅读及学习此章内容，肯定会有所收获。

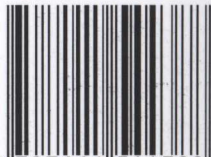


投稿热线: (010) 88379604
客服热线: (010) 88379426 88361066
购书热线: (010) 68326294 88379649 68995259

华章网站: www.hzbook.com
网上购书: www.china-pub.com
数字阅读: www.hzmedia.com.cn

上架指导: 计算机/操作系统/Linux

ISBN 978-7-111-54438-8



9 787111 544388 >

定价: 79.00元